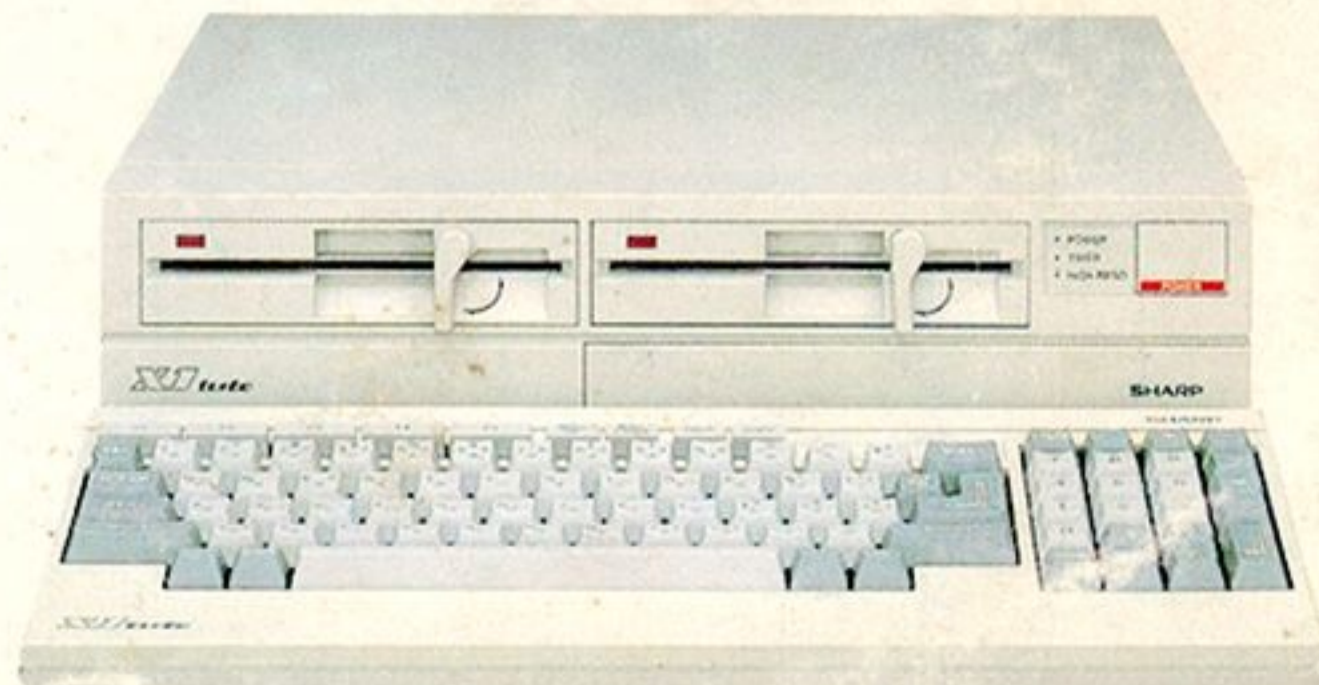
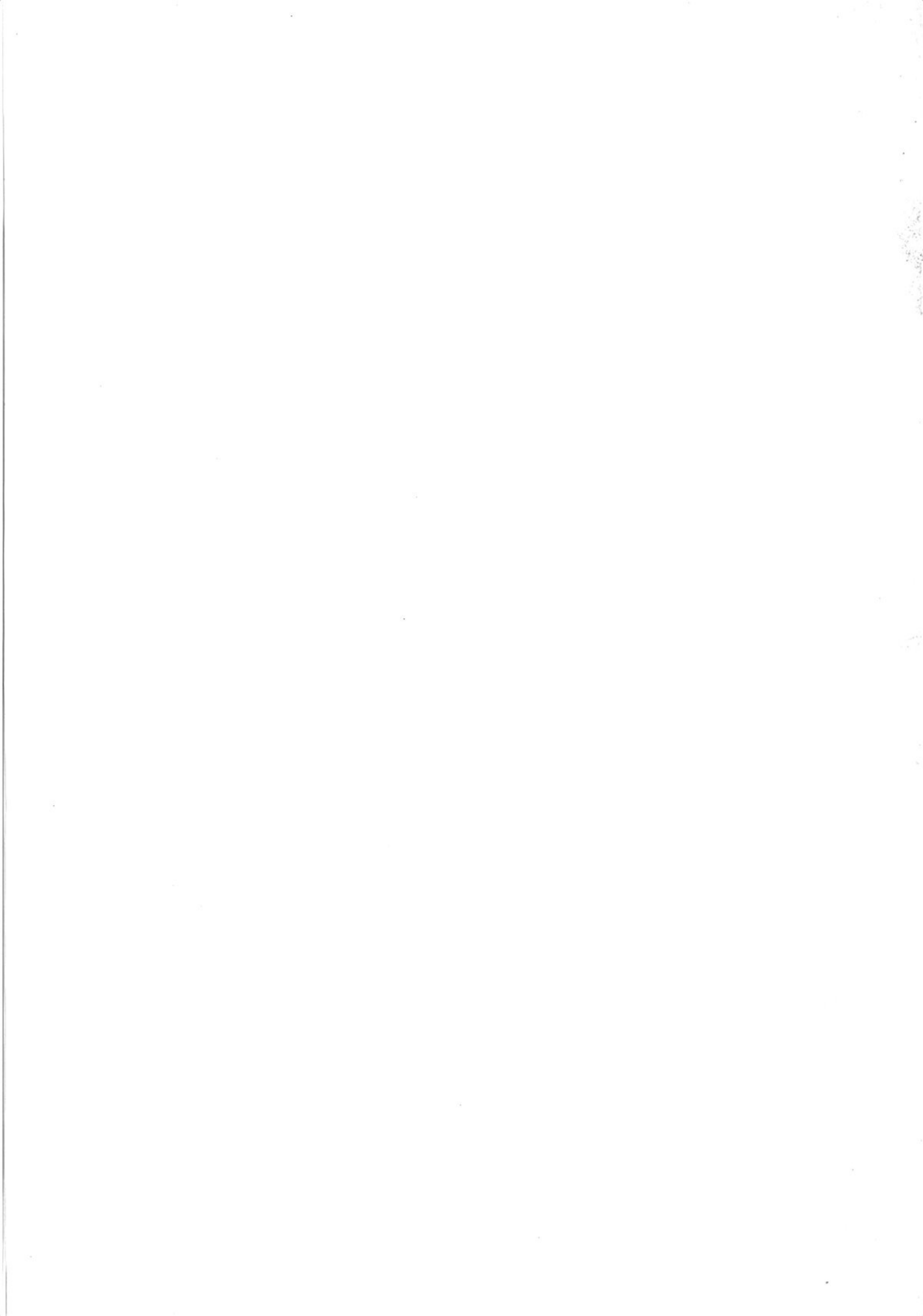
BASIC REFERENCE MANUAL

パソコンテレビX1turbo
パーソナルコンピュータ
形名
CZ-856C

上手に使って上手に節電

製造番号は、品質管理上重要なものですから商品本体に表示されている製造番号と、保証書に記載されている製造番号とが一致しているか、お確かめください。

はじめに

このシステムソフトウェアは、フロッピーディスクで供給しております。フロッピーディスクの取り扱いについては特に注意が必要ですので、取り扱い方法については、本機の取扱説明書を参照してください。

なお、マスターＢＡＳＩＣはコピーしておき、コピーのＢＡＳＩＣを使用されることをお奨めします。不慮の事故によってマスターＢＡＳＩＣが使用不能とならないよう安全な場所に保管ください。コピーをする場合は、『アプリケーションソフトの説明書』の「ディスクユーティリティ」をお読みください。

動作時のフロッピーディスクの入れ換えについてのご注意

フロッピーディスクをディスクドライブに挿入し、読み出し、書き込みを実行しているときは、フロッピーディスクの入れ換えを行なわないでください。

入れ換える必要がある場合には、フロッピーディスクの内容破壊防止のために、フロッピーディスクを入れ換える直前に、必ずＣＬＯＳＥ命令を実行してください。

ご注意

このマニュアルは、シャープパーソナルコンピュータの同梱ＢＡＳＩＣに基いて作成されています。

(1)システムソフトウェアは全てファイル形態のソフトウェアパック（フロッピーディスク）によってサポートされます。
各システムソフトウェアおよび本書の内容は、改良のため予告なく変更することがありますので、ファイルバージョンナンバーには、特にご注意されるよう、お願いいたします。

(2)システムソフトウェアならびに本書の内容を個人で使用する場合を除き、当社に無断で複製することは禁止します。

(3)本機は非常に複雑な機能および組合せを有する製品であり、出荷に際してマニュアルを含め十分なチェックをして万全を期しておりますが、万一ご使用中ご不審な点、お気付きのことがありましたら、もよりの「お客様ご相談窓口」までご連絡ください。
なお、運用した結果、生じる影響については、責任を負いかねますので、あらかじめご了承ください。

目次

1章　SHARP　HuBASICの基礎

2章　コマンド・ステートメント

3章 関数・システム変数

付録

アルファベット順索引

1章

SHARP HuBASICの基礎

この章では、BASICのプログラムを作るときに、最低限知っておきたい基本的な知識について説明しています。

1.1　ＢＡＳＩＣとは

機械語やアセンブラ言語といった計算機向きの言語は、あくまで計算機のための言語であるため、人間には判りにくいものとなっています。そこで、より人間が理解しやすいようにという目的で登場したのが、高水準言語です。

高水準言語には、ＢＡＳＩＣ、ＡＬＧＯＬ、ＦＯＲＴＲＡＮ、ＣＯＢＯＬ、ＬＩＳＰ、ＰＡＳＣＡＬなどがあります。

ＢＡＳＩＣは、Ｂｅｇｉｎｎｅｒ'ｓ　Ａｌｌ－ｐｕｒｐｏｓｅ　Ｓｙｍｂｏｌｉｃ　Ｉｎｓｔｒｕｃｔｉｏｎ　Ｃｏｄｅの略で、１９６４年、アメリカのダートマス大学で開発された高水準言語です。最初は教育用の言語として使用されていましたが、最近ではパソコンの発達に伴ない、その命令の数も飛躍的に増え、標準的な言語として広く知れ渡るようになりました。

ＳＨＡＲＰ　ＨｕＢＡＳＩＣは、本機の強力なハードウェアをサポートするために、特に開発されたＢＡＳＩＣです。

1.2　ＢＡＳＩＣプログラム

ＢＡＳＩＣを起動すると、ディスプレイテレビに、ＮＥＷＯＮ■と表示されますので、↵キーを押します。

　　Ｏｋ
　　■

と表示され、四角いカーソル[1] が明滅して、キーボードからの入力を待っています。

この状態のとき、キーボードのキーをタイプすると、画面のカーソルの位置にその文字が表示されます。

1）カーソル（ｃｕｒｓｏｒ）　画面表示において、次の文字を表示する位置を示すもので、中塗りの四角形■で表わされ、絶えず明滅（ブリンキング）しています。

1.2.1　ダイレクトモード

画面に、直接、ＢＡＳＩＣのコマンド[2] やステートメント[3] を入力して、↵キーを押すと、コンピュータはすぐ実行に取りかかります。

これをダイレクトモードでの実行といいます。

（例）ＰＲＩＮＴ"ＨＥＬＬＯ"↵（↵は、「ここで↵キーを押す」の意味です）
　　ＨＥＬＬＯ
　　Ｏｋ

2）コマンド（ｃｏｍｍａｎｄ）…「命令」という意味をもち、さまざまな使われ方をしますが、この説明書では、「ダイレクトモードで実行できる、プログラムやファイルに関する命令」という意味で用います。

3）ステートメント（ｓｔａｔｅｍｅｎｔ）…ここでは、「２５０以上あるＢＡＳＩＣの命令のうち、プログラムモードで実行できるもの」の意味で使っており、プログラムの命令文を構成する要素となっています。この説明書では、１つの命令文をステートメントと呼んでいます。詳しくは次の「プログラムモード」を参照してください。

1.2.2 プログラムモード

先頭に行番号をつけて、ステートメントを入力し、⏎キーを押すと、その行はコンピュータのメモリ[4)]に記憶されます。この行のことをプログラム行といい、プログラム行が1行以上集まったものをＢＡＳＩＣプログラム（または単にプログラム）といいます。

プログラムは、コンピュータが行なう一連の動作を指定する命令文から成り立っています。このプログラムを設計し、記述し、かつ試験することをプログラミングといいます。

メモリに記憶されたプログラムは、ＢＡＳＩＣのＲＵＮコマンドの入力により、実行することができます。

これをダイレクトモードでの実行に対し、プログラムモードでの実行といいます。

4）メモリ（ｍｅｍｏｒｙ）、… プログラムやデータを格納し、保持し、かつ取り出すことのできる記憶装置のこと。

プログラム行の記述形式は、次のようになっています。

行番号　ステートメント〔：ステートメント：……〕

（〔 〕内は省略できること、……は繰り返して書けることを意味します）

行番号：行番号は、プログラム行の順番を表わす１～６５５３４の整数で、通常は１０番おきにつけて、新しいプログラム行を行間に追加挿入するときに備えます。行番号は、順番を表わすほか、ＧＯＴＯ・ＧＯＳＵＢステートメントなどのジャンプ先の参照番号としての役割をもっています。プログラム行をコンピュータのメモリに記憶するには、先頭に必ず行番号をつけて、その行の上で⏎キーを押さなければなりません。

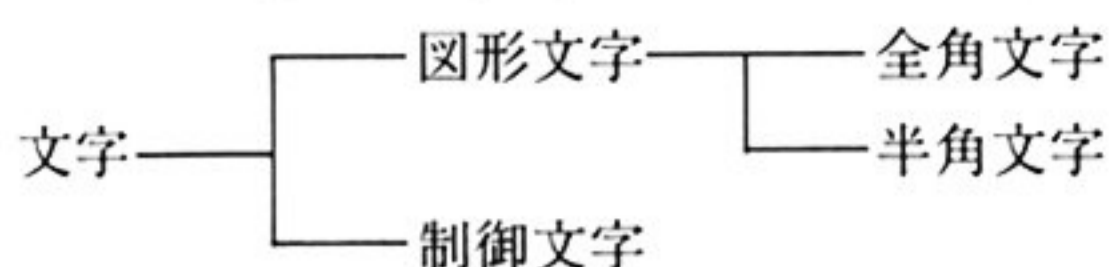
ステートメント：ステートメントは、ＢＡＳＩＣの各種のコマンド、ステートメント、関数、およびシステム変数によって記述される文で、コンピュータに実行させたい命令を表わしています。このステートメントは、コロン（：）で区切って、つなげて書くことができます。これを**マルチステートメント**と呼び、行番号を含めて２５５文字の長さまで入力することができます。また、行番号をつけずに、ダイレクトモードでも実行することができます。

プログラムは、原則として、行番号の順、書かれたステートメントの順に実行されますが、ステートメントの書き方次第で、前後にジャンプしたり、同じ部分を何度も繰り返し実行することができます。

（例1）
```
10  PRINT"HELLO"
20  END
```

（例2）
```
10  DIM X(10)
20  FOR I=1 TO 10
30  READ X(I):PRINT I,X(I)
40  NEXT
50  END
60  DATA 0,4,8,7,2,1,9,3,5,6
```
（例２の行番号３０はマルチステートメントになっています）

1.3 ＢＡＳＩＣで使われる文字

ＢＡＳＩＣで使われる文字（ｃｈａｒａｃｔｅｒ）は、次のように分類されます。

以下、この分類に従って説明します。

1.3.1 図形文字

図形文字（ｇｒａｐｈｉｃ　ｃｈａｒａｃｔｅｒ）は、画面に表示したりラインプリンタで紙に印刷して、目で見ることのできる記号で、英字などの外国文字、数字、漢字、カタカナ、ひらがな、空白（間隔文字）、セミグラフィック文字、および特殊文字があります。

英字（ａｌｐｈａｂｅｔｉｃａｌ　ｃｈａｒａｃｔｅｒ）はアルファベットの大文字（Ａ、Ｂ、Ｃ、………）と小文字（ａ、ｂ、ｃ、………）のことをいい、数字（ｄｉｇｉｔ）は０～９のアラビア数字のことをいいます。

また、セミグラフィック文字は「1.3.3　セミグラフィック文字」で、特殊文字は「1.3.4　特殊文字」で具体的に説明します。

なお、外国文字に対し、漢字、カタカナ、ひらがなをまとめて、日本語文字と呼ぶことがあります。

図形文字は、そのサイズによって、半角文字と全角文字の２つのタイプに分けられます。

半角文字（１バイトコード文字）

画面に表示したとき、カーソル１つ分のサイズをもつ文字で、英数字、カタカナ、セミグラフィック文字、その他特殊文字があります。

半角文字は１バイト[5]のメモリをとって記憶されるので、１バイトコード文字ともいいます。

半角文字は、ＢＡＳＩＣプログラム中で、定数[6]、変数名[7]、演算子[8]、および予約語[9]として用いることができます。

5）バイト（ｂｙｔｅ）　コンピュータ内部でデータを扱う際の基本となる単位で、８ビット（２進数８けた）を１バイトといいます。１バイトあれば、０～２５５のキャラクタコードを表わすことができます。したがって、文字型データの場合、１バイトで１つの半角文字を表わすことができます。

6）定数（ｎｕｍｅｒｉｃ）　「1.5　定数」参照。

7）変数名（ｖａｒｉａｂｌｅ　ｎａｍｅ）　「1.6　変数」参照。

8）演算子（ｏｐｅｒａｔｏｒ）　「1.9　演算子」参照。

9）予約語（ｒｅｓｅｒｖｅｄ　ｗｏｒｄ）　「1.4　予約語」参照。

全角文字（２バイトコード文字）

画面に表示したとき、カーソル２つ分のサイズをもつ文字で、英数字、カタカナ、セミグラフィック文字、漢字、ひらがな、記号があります。

全角文字は、２バイトのメモリをとって記憶されるので、２バイトコード文字ともいいます。

全角文字は、ＢＡＳＩＣプログラム中で、数値、演算子、および予約語として用いることができません。

1.3.2 制御文字

制御文字（control character）は、画面やプリンタを制御する命令として用いられ、特別なステートメントを使わなければ、画面に表示したり、プリンタで紙に印刷することはできません。

通常、目でみることができず、コンピュータ内では1バイトのコードとして取り扱われているので、コントロールコードともいいます。

コントロールコードについては、巻末の「コントロールコード」を参照してください。

1.3.3 セミグラフィック文字

セミグラフィック文字は、半角の文字で、次のようなものがあります。

▁ ▂ ▃ ▄ ▅ ▆ ▇ █ ▏ ▎ ▍ ▌ ▋ ▊ ▉ ╱
─ │ ┴ ┬ ┤ ├ ┼ ┐ ┘ └ ┌ ╮ ╰ ╯ ╭ ╲
● ○ ♠ ♥ ♦ ♣ ◢ ◣ ╳ ▘ ▖ ▝ ▗ ▞ ▚ □
░ 土 金 木 水 火 月 日 時 分 秒 年 円 人 生 〒

1.3.4 特殊文字

特殊文字は、半角の文字で、次のようなものがあります。

特殊文字	読み方	特殊な使われ方・働き
	空白	
!	感嘆符	単精度型属性文字
"	引用符	文字列を囲む記号
#	番号記号	ファイル番号記号、倍精度型属性文字
$	ドル記号	文字型属性文字
%	パーセント	整数型属性記号
&	アンパサンド	文字型書式指定子
'	アポストロフィ	注釈用区切り記号
(	左小かっこ	
)	右小かっこ	
*	アスタリスク	乗算記号
+	正符号、プラス	正符号、加算記号、文字列連結記号
,	カンマ	オペランドの区切り記号 (例) ＩＮＰＵＴ　Ａ，Ｂ 　　　ＰＲＩＮＴ　Ｘ，Ｙ，Ｚ 　　　ＤＡＴＡ　１０，１３，９１，４
−	負符号、マイナス	負符号、減算記号（＝ハイフン）
.	ピリオド	小数点として使うほか、ＢＡＳＩＣ内部の行番号管理ポインタとして使われます。プログラム実行中エラーの発生した行番号、ＳＴＯＰ・ＥＮＤステートメントによって停止した行番号、新しいプログラム行挿入時の行番号など、絶えず変化する行番号の値がセットされています。 (例) ＬＩＳＴ. 　　　ＥＤＩＴ. 　　　ＡＵＴＯ.
/	スラッシュ	除算記号
:	コロン	マルチステートメントの区切り記号
;	セミコロン	オペランドの区切り記号 (例) ＰＲＩＮＴ"コタエ＝"；Ａ 　　　ＰＲＩＮＴ　Ｘ；Ｙ；Ｚ 　　　ＩＮＰＵＴ"Ａ＝"；Ａ
<	不等号（より小）	
=	等号	
>	不等号（より大）	
?	疑問符	ＰＲＩＮＴステートメントの代用 (例) ？Ａ，Ｂ 　　　？ＦＲＥ（０） 　　　？ＴＩＭＥ＄
@	単価記号	
[	左大かっこ	
¥	円記号	整数除算記号

<table>
<tr><th>特殊文字</th><th>読　み　方</th><th>特 殊 な 使 わ れ 方 ・ 働 き</th></tr>
<tr><td>］</td><td>右大かっこ</td><td></td></tr>
<tr><td>＾</td><td>アクサンシルコンフレックス</td><td>累乗記号</td></tr>
<tr><td>＿</td><td>アンダーライン</td><td></td></tr>
<tr><td>｀</td><td>アクサングラーブ</td><td></td></tr>
<tr><td>｛</td><td>左中かっこ</td><td></td></tr>
<tr><td>|</td><td>縦線</td><td></td></tr>
<tr><td>｝</td><td>右中かっこ</td><td></td></tr>
<tr><td>￣</td><td>オーバーライン</td><td></td></tr>
<tr><td>－</td><td>ハイフン</td><td>ＬＩＳＴ・ＤＥＬＥＴＥステートメントなどで行の範囲を指定したりするときに使います。
(例) ＬＩＳＴ　１００－３００
　　 ＤＥＬＥＴＥ　１０００－３０００
　　 ＤＥＦＤＢＬ　Ａ－Ｈ</td></tr>
</table>

1.4 予約語

予約語（ｒｅｓｅｒｖｅｄ　ｗｏｒｄ）は、ＢＡＳＩＣインタープリタが管理している語で、その意味が特定の規則によって固定されており、プログラム中で変数名、ユーザー定義の関数名として使うことはできません。

すべてのコマンド、ステートメント、関数、演算子およびシステム変数が予約語になっています。その他下にも示す語も予約語になっています。

ＴＯ　ＳＴＥＰ　ＴＨＥＮ　ＵＳＩＮＧ　ＳＵＢ　ＢＡＳＥ　ＴＡＢ　ＳＰＣ
ＤＥＦ　ＯＮ　ＯＦＦ　ＥＬＳＥ

1.5 定数

定数とは、ＢＡＳＩＣで使われる値で、数値定数と文字定数の２つの種類があります。

1.5.1 数値定数

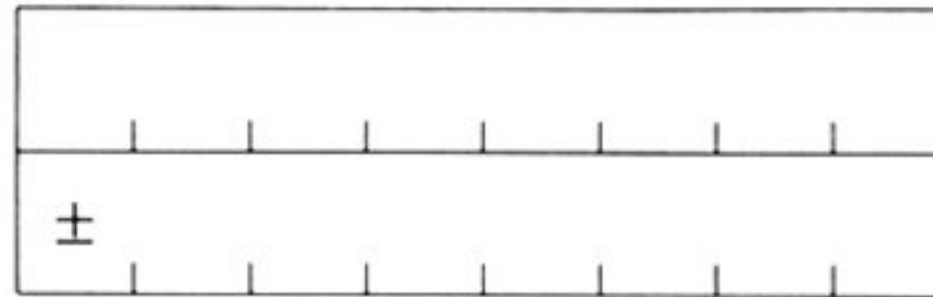
数値定数は、正または負の数値で０も含みます。表示形式の違いから、１０進数、１６進数、８進数、２進数、ＪＩＳ１６進定数、ＪＩＳ区点定数の６つの種類に分けられます。

１０進数

１０進数は、０～９の数字および小数点（．）で表わされる数で、日常使われているものです。正と負の区別をするのに正符号（＋）と負符号（－）を使いますが、＋は省略しても使えます。整数型と実数型の２つのタイプがあります。

〔1〕整数型

整数型は、－３２７６８以上＋３２７６７以下の整数で、ＢＡＳＩＣ内部では2バイト（１６ビット）のメモリをとって記憶されています。

整数型………２バイト

（例）４１５０
２３５
－４４０

〔２〕実数型

実数型には、単精度型と倍精度型の２つのタイプがあります。

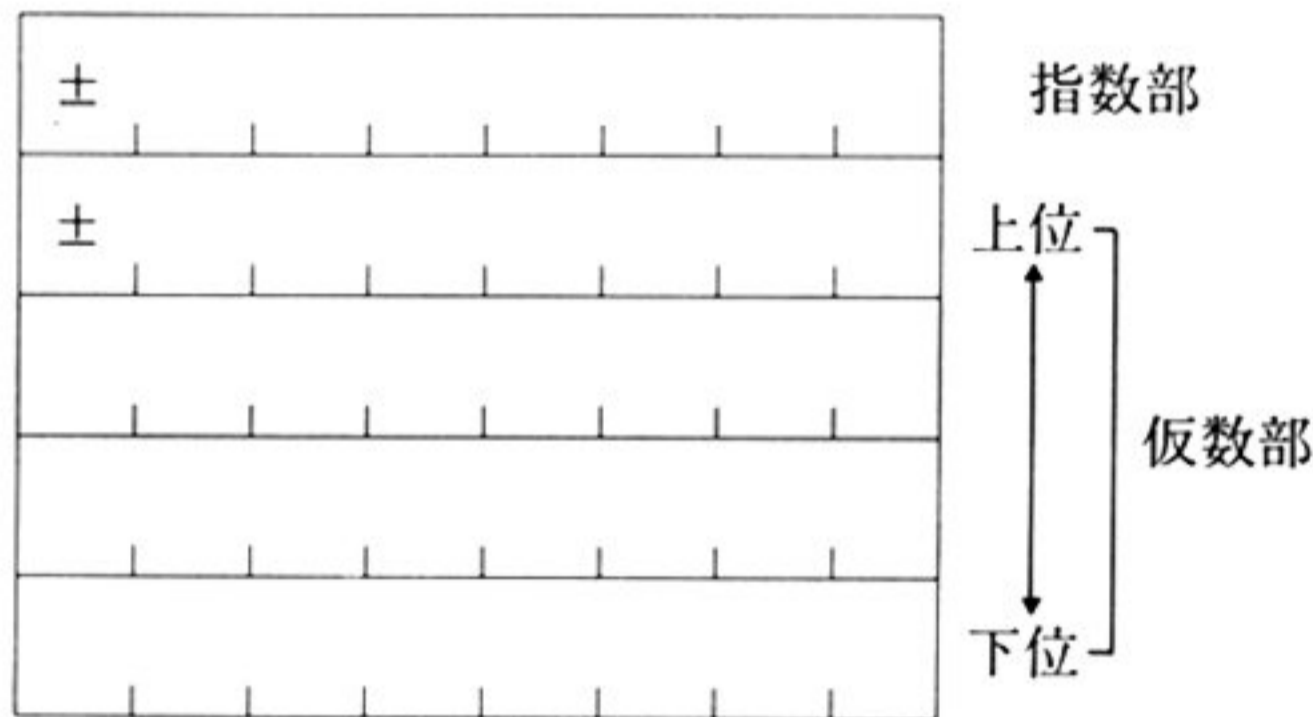
（１）単精度型

単精度型は、表示けた数が８けた以下で約－１．７×１０^{38}以上＋１．７×１０^{38}以下（指数は１０$^{-38}$～１０^{38}）の範囲の数値です。

けた数が、８けた以下のときは、１０進記数法[1]で表わされ、８けたを越えると浮動小数点法[2]で表わされます。

浮動小数点表示法は、＊．＊＊＊＊＊＊＊Ｅ±ｎの形式で、仮数部（＊．＊＊＊＊＊＊＊の部分）が－１．７０１４１１８以上１．７０１４１１８以下の８けたまで、指数部（±ｎの部分）が－３８以上＋３８以下の範囲で表わされます。

ＢＡＳＩＣ内部では、指数部が１バイト（８ビット）、仮数部が４バイト（３２ビット）合計５バイトのメモリをとって記憶されています。

単精度型実数………５バイト

１）１０進記数法……表示けた数（単精度型８けた、倍精度型１６けた）以内で数を表示する形式で、普通の１０進数表現のことです。小数点以下の有効けた数をはみ出した右側の部分は四捨五入されます。

（例）２３５．３
３．１４１５９２７
５３２０００００

２）浮動小数点表示法……表示けた数より大きい数を表示する形式で、指数形式ともいいます。－２３５０００００００を－２．３５×１０^{8}と表示して使うように、ＢＡＳＩＣでは－２．３５Ｅ＋８と表示して、小数点の位置を指示する数字８を後ろに併記します。このとき、－２．３５の部分を仮数部、＋８の部分を指数部と呼びます。

（例）２．３５３Ｅ＋２
－９．３５Ｅ＋９
１．７０１４１１８Ｅ＋３８

（２）倍精度型

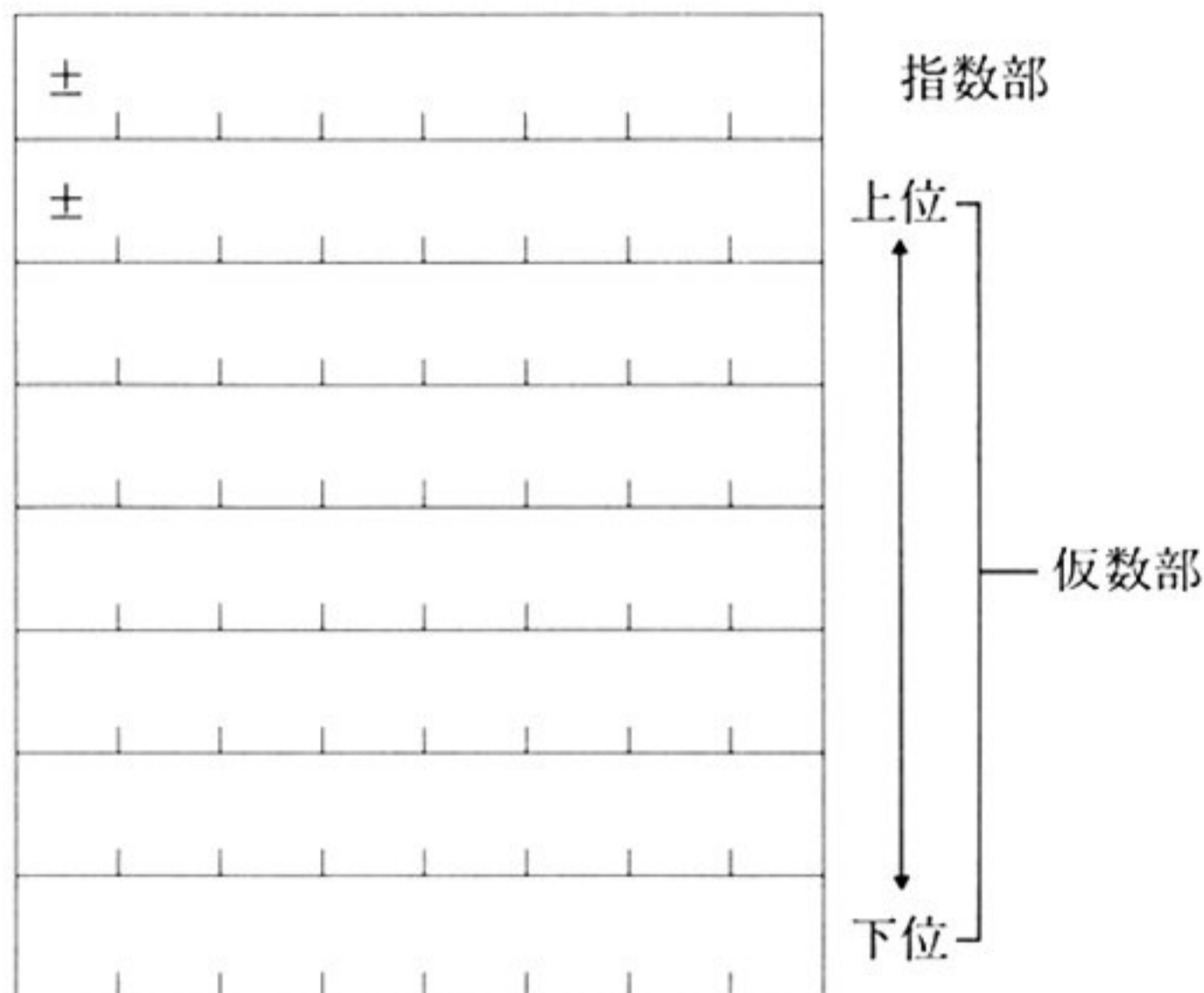
倍精度型は、表示けた数が１６けた以下で、取り扱える数の範囲は単精度の場合と同じです。

けた数が１６けた以下のときは、１０進記数法で表わされ、１６けたを越えると浮動小数点表示法で表わされます。

浮動小数点は、＊．＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊Ｄ±ｎの形式で、仮数部が１６けた以下、指数部が－３８以上＋３８以下の範囲で表わされます。

定数が倍精度数であることを表わす記号に番号記号（#）があり、数字の後ろにつけて表わします。ただし、8けたを越える場合は、#を省略しても自動的に倍精度型とみなされます（『ユーザーズマニュアル』・付録「数値精度の変換」参照）。

ＢＡＳＩＣ内部では、指数部が1バイト（８ビット）、仮数部が7バイト（５６ビット）合計8バイトのメモリをとって記憶されています。

倍精度型実数………８バイト

（例）１０進記数法

　　１９８４#

　　１．７３２#

　　３．１４１５９２６５３５８９７９３

　浮動小数点表示法

　　５．６Ｄ＋９

　　－１．２３４５６Ｄ＋１８

　　１．２３４５６７８９０１２３４５６Ｄ－３８

１６進数

１６進数は、ある位が１６になると１つ上位に上がる数で、０～９の数字と、数字の代用としてＡ～Ｆの英字で表わします。

数値は、ＢＡＳＩＣ内部では２進数で表現されていますが、２進数を直接読んだり書いたりするのは、けたが大きくて不便です。そこで、次のように２進数の４けたを１けたで表現できる１６進数を使用します。表現は違っても、ＢＡＳＩＣ内部では整数と全く同じもので、2バイト（１６ビット）のメモリをとって記憶されています。

（例）２進数表現：　<u>１１００</u>　<u>０１１０</u>，<u>０１００</u>　<u>０１１０</u>，<u>０１００</u>　<u>１１１１</u>

　　１６進数表現：　Ｃ　６　４　６　４　Ｆ

１６進数は、１０進数と区別するため（Ｃ６）$_{16}$のように表現することがありますが、ＢＡＳＩＣでは１６進数の頭に＆Ｈをつけて表わします。

１６進数で表現できる数の範囲は、＆Ｈ０～＆ＨＦＦＦＦです。

（例）＆ＨＣ６

＆Ｈ４Ｆ０

＆ＨＣ６４Ｆ　　（ＨはＨｅｘａの意味です）

８進数

８進数は、ある位が８になると１つ上位に上がる数で、０～７の数字で表わされます。

これも、１６進数と同じ理由で、２進数の代りとして用いられ、数字の頭に＆Ｏ０をつけて表わします。表現できる数の範囲は＆Ｏ０～＆Ｏ１７７７７７です。表現は違っても、ＢＡＳＩＣ内部では整数と全く同じもので、２バイト（１６ビット）のメモリをとって記憶されています。

（例）＆Ｏ３０６

＆Ｏ２３６０

＆Ｏ１４３１１７　　（ＯはＯｃｔａｌの意味です）

２進数

２進数は、０と１の２つの数字で表わされる数です。２進数は、１０進数と区別するため（１１０００１１０）$_2$ のように表現することがありますが、ＢＡＳＩＣでは２進数の頭に＆Ｂをつけて表わします。

表現できる数の範囲は、＆Ｂ０～＆Ｂ１１１１１１１１１１１１１１１１の１６けたです。表現は違っても、ＢＡＳＩＣ内部では整数と全く同じもので、２バイト（１６ビット）のメモリをとって記憶されています。

（例）＆Ｂ１１０００１１０

＆Ｂ１００１１１００００

＆Ｂ１１０００１１００１００１１１１　　（ＢはＢｉｎａｒｙの意味です）

ＪＩＳ１６進定数

ＪＩＳ１.６進定数は、ＪＩＳ１６進コード[1)]の頭に＆Ｊをつけたもので、ＪＩＳ１６進コードに対応するシフトＪＩＳコード[2)]の値をもちます。

ＪＩＳ１６進コードは、１６進数４けたで表わされ、上位２けたと下位２けたがそれぞれ２１～７Ｅの範囲の１６進数です。

（例）＆Ｊ２１２１　…………　ＪＩＳ１６進コードは２１２１で、これに対応するシフトＪＩＳコードは＆Ｈ８１４０なので、＆Ｊ２１２１は＆Ｈ８１４０の値をもちます。

↑

ＪＩＳ１６進コード

１）ＪＩＳ１６進コード　付録「コード体系」参照。

２）シフトＪＩＳコード　付録「コード体系」参照。

ＪＩＳ区点定数

ＪＩＳ区点定数は、区点コード[3)]の頭に＆Ｋをつけたもので、区点コードに対応するシフトＪＩＳコードの値をもちます。

区点コードは、１０進数４けたで表わされ、上位２けたと下位２けたがそれぞれ０１～９４の範囲の１０進数です。

（例）＆Ｋ０１０１　…………　区点コードは０１０１で、これに対応するシフトＪＩＳコードは＆Ｈ８１４０なので、＆Ｋ０１０１は＆Ｈ８１４０の値をもちます。

↑

区点コード

３）区点コード　付録「コード体系」参照。

1.5.2　文字定数

文字定数は、次のように文字列を引用符（"）で囲んだもので、単に文字列ともいいます。

(例)　"地球"

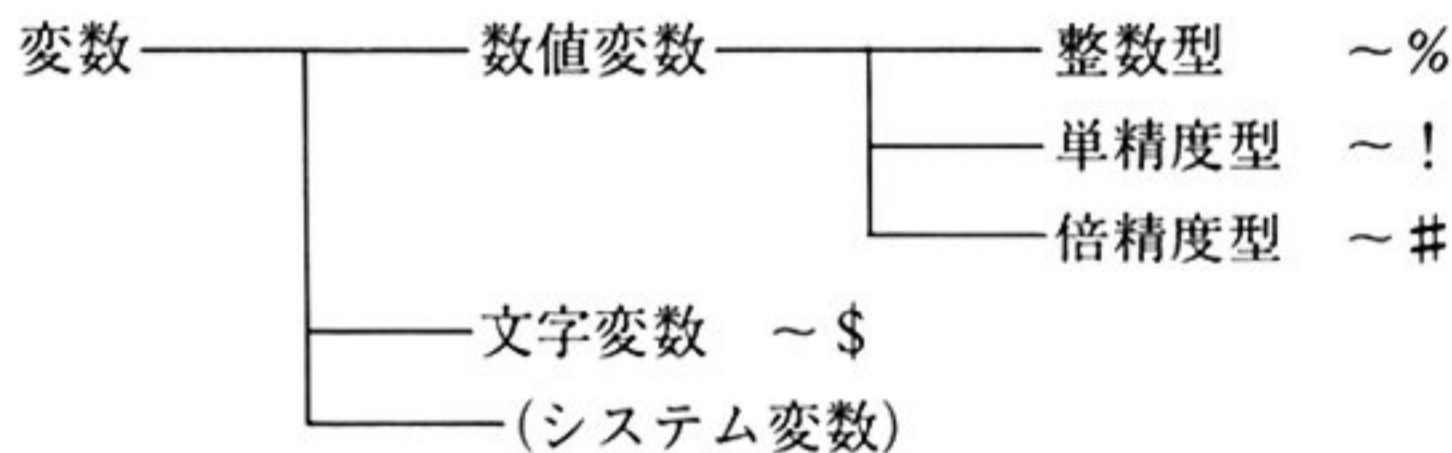
"２００１／０９／１５"

"パソコンテレビ"

文字列の長さは２５５文字以内に制限されています。引用符は文字列の中に含めることができません。

長さが０の文字列をヌルストリングと呼び、引用符を２つ並べて、""と表わします。

1.6　変数

変数とは、一口に言うと、定数を入れておく入れ物のことで、定数の型に対応して次のように分けられます。

変数の別を識別するために、変数名の後ろに「%、!、#、$」の属性文字とよばれる記号をつけます。

変数に値を入れないで使うと、数値変数は０、文字変数はヌルストリング（長さ０で文字が入っていない）とみなされます。

変数は、変数名によって識別することができます。変数名をつけるとき注意することは次の３つの点です。

(a)変数名としてつかわれる文字の先頭の１文字目は必ず英字もしくは［でなければなりません。
　２文字目以降は、英字・数字のどちらでも使えます。なお、間に属性文字（%、!、#、$）を入れてはいけません。
　先頭に［を使用した場合は、変数名の最後は］でくくらなければなりません。［から］の間は全角文字も含め、あらゆる文字が使用できます。

(誤)　５Ａ　Ａ%Ｂ　Ｘ#Ｙ

(正)　Ａ５　ＡＢ%　ＸＹ#

(b) 変数名の長さは、最大２４０バイトです。

(c) 変数名の先頭に予約語を入れてはいけません。

(誤)　ＲＵＮＡ　ＦＯＲＡＩ　ＳＩＮＸ２

(正)　ＡＲＵＮ　ＡＦＯＲＩ　Ｘ２ＳＩＮ

1.6.1　数値変数

数値変数には、整数型、単精度型、倍精度型の３つのタイプがあります。

整数型

整数型の数値変数は、変数名の後ろに%をつけて表わします。この型の変数に入れておくことのできる定数は、－３２７６８～３２７６７の整数です。１つの整数を記憶するには２バイトのメモ

リが必要です。

（例）Ａ％
　　　Ｈｕ％
　　　ＳＥＩＳＵ％
　　　［漢字］％

単精度型

単精度型の数値変数は、変数名の後ろに！をつけて表わしますが、通常は省略して使えます。属性文字のついていない変数は、後述の型宣言をしていない限り、すべて単精度型とみなされます。入れることのできる数値は、単精度型の実数です。

１つの単精度型実数を記憶するには５バイトのメモリが必要です。

（例）Ａ　　　Ａ！
　　　Ｈｕ　　Ｈｕ！
　　　ＳＵＵ　ＳＵＵ！
　　　［漢字］！

倍精度型

倍精度型の数値変数は、属性文字のついていない変数のうち［ではじまる変数は、後述の型宣言をしていない限り、すべて倍精度型とみなされます。変数名の後ろに＃をつけて表わします。入れることのできる数値は倍精度型の実数です。

１つの倍精度型実数を記憶するには８バイトのメモリが必要です。

（例）Ａ＃
　　　Ｈｕ＃
　　　ＢＡＩ＃
　　　［漢字］＃

1.6.2　文字変数

文字列（文字定数）を入れることのできる変数を文字変数といい、変数名の後ろに＄をつけて表わします。

文字変数に入れる文字列の長さは２５５バイトまでです。

（例）Ａ＄
　　　Ｈｕ＄
　　　ＭＯＪＩ＄
　　　［漢字］＄

1.6.3　システム変数

システム変数には、刻々と変化するＢＡＳＩＣ内部の状態を表わす値が入っています。

システム変数は、次の４つに分類できます。

それぞれの詳しい説明は、「関数・システム変数」を参照してください。

1.7 配列

配列とは、1つの変数名によって、一連のデータを表わすコンピュータの情報処理における工夫です。

配列をプログラム中で使うには、初めにDIMステートメントで、

```
10 DIM A(3)
```

のように定義します。

DIMは、配列を定義するステートメントで、配列を定義することを配列宣言といいます。「DIM A(3)」の「A」を配列名、( )の中の数字を配列のサイズといいます。

この場合、A(0)、A(1)、A(2)、A(3)の4つの変数の領域が、メモリに確保されます。ここで、( )の中に書いてある0～3の数字を添字といい、こと添字の値によって変数の格納場所が指定されます。

配列名のつけ方は変数名のつけ方に従います。

配列のサイズは、カンマで区切って、いくつか並べて書くことができます。1つのときは1次元配列、2つのときは2次元配列、3つのときは3次元配列、……といいます。

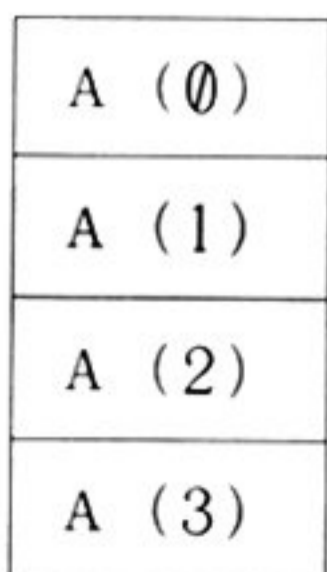
(例) DIM A(3) ………………… 1次元配列
DIM A(3, 3) ………… 2次元配列
DIM A(3, 3, 3) …… 3次元配列

配列のサイズおよび添字は定数のほか、変数、式のいずれで指定しても使えます。
ただし、0以上の整数を値にもつものでなければなりません。

(例) A(I)
A(2*X)
A(MAX)

最初の例のようにDIM A(3)と宣言することは、次の図のように変数を一列に並べることと考えることができます。

1次元配列

A(0)
A(1)
A(2)
A(3)

このように考えると、2次元配列を

```
10 DIM A(2, 3)
```

と宣言するには、次のような格子を作るようなもので、横方向Ｊと縦方向Ｉの２つのパラメータによって１つの場所が定まると考えることができます。

２次元配列

列　Ｊ

行 Ｉ

A(0，0)	A(0，1)	A(0，2)	A(0，3)
A(1，0)	A(1，1)	A(1，2)	A(1，3)
A(2，0)	A(2，1)	A(2，2)	A(2，3)

＜２次元配列のメモリ内表現＞

A（0，0）
A（1，0）
A（2，0）
A（3，0）
A（0，1）
A（1，1）
⋮
A（2，3）

1.8 代入

変数や配列に定数や式の値を入れることを「代入する」といいます。

代入の記号としては、等号（＝）が使われます。この等号は、「両辺が等しい」という意味ではなく、「右辺の値を左辺に代入する」という意味で用い、右辺には代入したい式、変数および定数、左辺には代入先の変数や配列が置かれます。このとき両辺の型は一致していなければなりません。

（例）Ａ％＝５……整数型変数Ａ％の値は５になります。

Ｃ＝Ａ＋Ｂ……Ａの値が１．２、Ｂの値が２．３のとき、単精度型変数Ｃの値は３．５になります。

変数の値は、代入によって何度でも書き換えることができます。

（例）Ａ％＝１０……Ａ％の値が５のとき、値は１０に変わります。

左辺の変数名が右辺の式に現われても行なえます。

（例）Ａ％＝Ａ％＋１……Ａ％の値が１０のとき、まずＡ％＋１が計算され、１０＋１が左辺のＡ％に代入されます。よって、Ａ％の値は１１になります。

文字型変数への文字定数（文字列）の代入も、数値の代入と同様に行なうことができます。

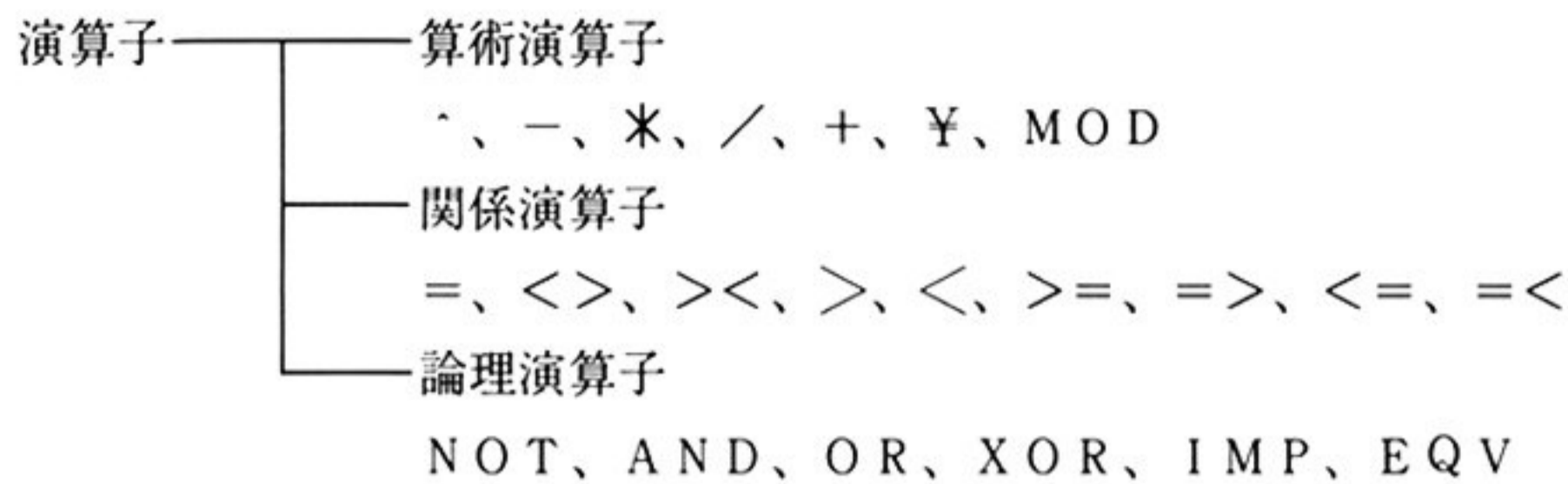
（例）ＭＯＪＩ＄＝"ＨＥＬＬＯ"……文字変数ＭＯＪＩ＄の値は、ＨＥＬＬＯとなります。

ＭＯＪＩ＄＝ＭＯＪＩ＄＋"、ＦＲＩＥＮＤＳ！"……ＭＯＪＩ＄の値がＨＥＬＬＯのとき、左辺のＭＯＪＩ＄の値は、ＨＥＬＬＯ、ＦＲＩＥＮＤＳ！となります。

1.9 演算子

演算（計算）に使われる＋、－などの記号のことを演算子（ｏｐｅｒａｔｏｒ）といいます。

これに対して、演算が行なわれる対象となる数値や文字列をオペランド（ｏｐｅｒａｎｄ）といいます。(広義には、ＢＡＳＩＣの命令中、処理の対象となるデータをすべてオペランドといいます)。

演算子には、算術演算子、関係演算子、および論理演算子の3種類があります。

1.9.1 算術演算子

算術演算子（arithmetic operator）には、次の7つがあります。

算術演算子	演算の内容	例	数学の表現
^	累　乗	X ^ 3	X^3
*	乗　算	A*B	AB
／	除　算	A／B	A÷B
＋	加算または符号	A＋B、＋A	A＋B、＋A
−	減算または符号	A−B、−A	A−B、−A
¥	整数除算	A¥B	
MOD	剰余の計算	A　MOD　B	

演算子^、*、／、¥、およびMODは、それぞれ2つのオペランドの間に書き、そのオペランドの対に作用します。

演算子＋および−は、2つのオペランドの間に書き、そのオペランドの対に作用するか、1つのオペランドの前に書き、そのオペランドに作用するか、のいずれかです。

¥は整数除算の演算子で、これを使って除算すると、結果（商）が小数点以下の切り捨てられた整数（整数化した商）になります。

(例) 10.5／3……商は3.5です。

　　10.5¥3……商の小数点以下.5が切り捨てられ、結果が3になります。

MODは、除算をしたときの余りを計算します。

(例) 13　MOD　5……13÷5＝2　余り3　なので、結果は3になります。

数学において、2つ以上の演算子を含む四則計算をするときは、まず累乗を、続いて乗除算を、最後に加減算をするというように、計算する順序が決まっていますが、BASICの算術式の演算にも同様の規則があり、オペランドの結合順序はかっこによる順序の変更がない限り、次の優先順位に従います。

優先順位	算術演算子
1.	^
2.	+、−（符号）
3.	＊、／
4.	￥
5.	MOD
6.	+、−（加減）

演算順位が等しい場合は、式の左側の演算子から先に演算されます。

演算の順序を変える場合は、先にしたい演算をかっこ（　）で囲みます。

(例)〔BASICの表現〕	〔数学の表現〕
A＊X＋Y	$AX+Y$
A＊X^2−B＊X＋Y	AX^2-BX+Y
(X＋Y)／2	$\frac{X+Y}{2}$
X＋Y／2	$X+\frac{Y}{2}$
(−X)^2	$(-X)^2$
X^−2	X^{-2}
−X^2	$-X^2$
(X^Y)^2	$(X^Y)^2$
X^(2＊Y)	X^{2Y}
X^(Y^2)	X^{Y^2}

1.9.2 関係演算子

関係演算子（relational operator）は、2つのオペランドの間に書き、両オペランドの値を比較するときに使います。

オペランドは、数式、または文字式です。数式には、単独の定数、数値変数、関数も含め、文字式は、単独の文字列、文字変数、関数およびそれらを連結したものです。

2つのオペランドの値が演算子によって指定された関係を満足すれば、真の値−1をもち、満足しなければ、偽の値0をもちます。

次に関係演算子を示します。

関係演算子	意　　味
＝	両辺が等しい（＝）
<>、><	両辺が等しくない（≠）
>	左辺が右辺より大きい
<	左辺が右辺より小さい
>=、=>	左辺が右辺より大きいか等しい（≧）
<=、=<	左辺が右辺より小さいか等しい（≦）

関係演算子は、プログラムのＩＦステートメントの中で、条件式としてよく使われます。
(例) １００　ＩＦ　Ｘ＞０　ＴＨＥＮ　２００……「変数Ｘの値が０より大きければ、２００番にジャンプしなさい」という意味になります。
詳しくは、ＩＦ－ＴＨＥＮ－ＥＬＥＳステートメントの項を参照してください。
また、Ａ＝Ｘ＞０あるいは　Ａ＝（Ｘ＞０）のように使うと、Ｘが０より大きいか否かを判断することができます。

1.9.3 論理演算子

数値は、ＢＡＳＩＣ内部では０と１をつかった２進数で表現されています。論理演算子（ｌｏｇｉｃａｌ　ｏｐｅｒａｔｏｒ）は、２つのオペランドの間に書き、両オペランドのビット単位でのブール演算を行なう記号で、次の６つがあります。

演算順位	論理演算子	意　　味	
1	ＮＯＴ	ｎｏｔ	否定
2	ＡＮＤ	ａｎｄ	論理積
3	ＯＲ	ｉｎｃｌｕｓｉｖｅ　ｏｒ	論理和
4	ＸＯＲ	ｅｘｃｌｕｓｉｖｅ　ｏｒ	排他的論理和
5	ＩＭＰ	ｉｍｐｌｉｃａｔｅ　ｉｎ	包含
6	ＥＱＶ	ｅｑｕｉｖａｌｅｎｔ　ｔｏ	同値

ブール演算（Ｂｏｏｌｅａｎ　ｏｐｅｒａｔｉｏｎ）とは、数学のブール代数の規則に従う演算で、真を１、偽を０とする２つの値の間で行なわれ、その結果（真理値）も、１（真）か０（偽）の２つの値で表わされる演算です。
次に、上の２つの論理演算子それぞれについて、具体的な演算内容を直理値表（Ｔｒｕｔｈ　ｔａｂｌｅ）としてまとめます。

ＮＯＴ（否定）

Ｘ	ＮＯＴ　Ｘ
1	0
0	1

＊これは、Ｘが１のとき、「ＮＯＴ　Ｘ」が０になり、Ｘが０のとき「ＮＯＴ　Ｘ」が１になることを表わしています。

ＡＮＤ（論理積）

Ｘ	Ｙ	Ｘ　ＡＮＤ　Ｙ
1	1	1
1	0	0
0	1	0
0	0	0

＊これは、Ｘが１かつＹが１のときのみ、「Ｘ　ＡＮＤ　Ｙ」が１になり、それ以外は0になることを示しています。これはちょうど、ＸとＹの乗算とその結果（積）に対応しています。

ＯＲ（論理和）

Ｘ	Ｙ	Ｘ　ＯＲ　Ｙ
1	1	1
1	0	1
0	1	1
0	0	0

＊これはちょうど、ＸとＹの加算とその結果（和）に類似しています。

ＸＯＲ（排他的論理和）

Ｘ	Ｙ	Ｘ　ＸＯＲ　Ｙ
1	1	0
1	0	1
0	1	1
0	0	0

＊排他的というのは、ＸとＹとが違うときに１（真）となると定めたことによります。

ＩＭＰ（包含）

Ｘ	Ｙ	Ｘ　ＩＭＰ　Ｙ
1	1	1
1	0	0
0	1	1
0	0	1

＊これは、Ｘ≦Ｙの比較とその結果に対応しています。

ＥＱＶ（同値）

Ｘ	Ｙ	Ｘ　ＥＱＶ　Ｙ
1	1	1
1	0	0
0	1	0
0	0	1

＊ＸとＹの値が同じとき１、違うとき0となります。

以上がブール演算の基本となる操作です。

次に、論理演算子が具体的にどのような働きをするのかみてみましょう。

Ａ＝１２　ＡＮＤ　１０

としたとき、１２と１０はそれぞれ２進法の１６ビットの値として表現され、各ビットごとにＡＮＤ（論理積）の演算が行なわれます。

```
12  =  (0000 0000 0000 1100)2
        |||| |||| |||| ||||
       ----------------------
               A N D
       ----------------------
        |||| |||| |||| ||||
10  =  (0000 0000 0000 1010)2
        ↓↓↓↓ ↓↓↓↓ ↓↓↓↓ ↓↓↓↓
       (0000 0000 0000 1000)2  =  8
```

したがって、１２　ＡＮＤ　１０は、２進法の０００００　００００　００００　１０００になり、Ａの値は８になります。

それでは、

Ａ＝Ｂ＞０　ＯＲ　Ｃ＞０

という式で、Ｂ＞０が成り立ち、Ｃ＞０が成り立たないとすると、Ａはどのような値になるでしょうか。成り立つとは真であることなので－１、成り立たないことは偽であることなので０の値をもちます。

Ａ＝－１　ＯＲ　０

－１は、１のＮＯＴをとって１を加えた値（１の２の補数）として表現されますから、

１＝（００００　００００　００００　０００１）$_2$

ＮＯＴをとると、

（１１１１　１１１１　１１１１　１１１０）$_2$

１を加えて、

（１１１１　１１１１　１１１１　１１１１）$_2$

よって、－１は（１１１１　１１１１　１１１１　１１１１）$_2$ と表現されるので、－１　ＯＲ　１の演算は次のように行なわれます。

```
-1  =  (1111 1111 1111 1111)2
        |||| |||| |||| ||||
       ----------------------
                O R
       ----------------------
        |||| |||| |||| ||||
 0  =  (0000 0000 0000 0000)2
        ↓↓↓↓ ↓↓↓↓ ↓↓↓↓ ↓↓↓↓
       (1111 1111 1111 1111)2  =  -1
```

よって、－１　ＯＲ　０は、２進法の１１１１　１１１１　１１１１　１１１１になり、Ａの値は－１になります。

このように、論理演算は、２進法１６ビットの数値表現によって行なわれるので、演算できる数値は、－３２７６８～３２７６７の範囲でなければなりません。この範囲をはみ出すと「Ｏｖｅｒｆｌｏｗ」のエラーが出ます。

論理演算子を使って、数値のビットレベルでの簡単な操作をすることができます。

（例）Ｘ＝Ｘ　ＡＮＤ　７……こうすると、右辺のＸの値にかかわらず、左辺のＸは０～７になります。

論理演算子は、（しばしば関係演算子を伴った）条件式としてＩＦステートメントの中で用いて、プログラムの流れを変えるのに使います。

(例) １００　IF　X<０　OR　X>５　THEN　２００……「変数Xの値が、０より小さいが５より大きいならば、２００番にジャンプしなさい」という意味になります。

1.9.4 演算子の優先順位

これまで述べてきた３種類の演算子には、次のような優先順位があり、混合して使うことができます。

1．算術演算子
2．関係演算子
3．論理演算子

すべての演算子の優先順位をまとめると次のようになります。

1．（　）で囲まれた部分の式
2．関数
3．＾（累乗）
4．＋、－（プラス・マイナス符号）
5．＊、／（剰除算）
6．￥　　（整数除算）
7．ＭＯＤ（剰余計算）
8．＋、－（加減算）
9．＝、＜＞、＞＜、＞、＜、＞＝、＝＞、＜＝、＝＜　（関係演算子）
10．ＮＯＴ（否定）
11．ＡＮＤ（論理積）
12．ＯＲ（論理和）
13．ＸＯＲ（排他的論理和）
14．ＩＭＰ（包含）
15．ＥＱＶ（同値）

1.9.5 文字列の演算

数値の演算と同様に、演算子を用いて文字列の演算を行なうことができます。文字列の演算には、連結と比較の２種類あります。

文字列の連結

２つ以上の文字列（文字定数に同じ）は、演算子の＋を用いて、より長い１つの文字列にまとめることができます。

(例) Ａ＄＝"＆Ｈ"＋"Ｃ０ＦＦ"……文字変数Ａ＄は、＆ＨＣ０ＦＦとなります。

連結に使う＋は、算術演算子の＋とは少し意味が違い、「文字列と文字列をつなげる」という意味をもっています。

文字列と文字変数、文字変数と文字変数をつなげることも可能です。

(例) Ａ＄＝"＆Ｈ"＋Ｘ＄…Ｘ＄にＣ０ＦＦが入っていると、Ａ＄は＆ＨＣ０ＦＦとなります。
　Ｆ＄＝"ＣＡＳ："＋ＦＬ＄……ＦＬ＄に"ＴＥＳＴ"が入っていると、Ｆ＄は"ＣＡＳ：ＴＥＳＴ"となります。
　Ｆ＄＝ＦＤ＄＋ＦＬ＄……ＦＤ＄に"ＭＥＭ："、ＦＬ＄に"ＴＥＳＴ"が入っていると、Ｆ＄は"ＭＥＭ：ＴＥＳＴ"となります。

なお、文字列の演算で－を使って削除はできません。すなわち、文字列ＡＢＣＤＥからＤＥを削除した結果ＡＢＣをＸ＄に入れようとして、

　　Ｘ＄＝"ＡＢＣＤＥ"－"ＤＥ"

としても、「Ｔｙｐｅ　ｍｉｓｍａｔｃｈ」のエラーが出てしまいます。この場合、次のようにＬＥＦＴ＄関数を使えば正しい結果を得ることができます。

　　Ｘ＄＝ＬＥＦＴ＄（"ＡＢＣＤＥ"，３）……文字列ＡＢＣＤＥの左から３文字取り出し、
　　　　Ｘ＄にはＡＢＣが入ります。

同様に、文字列の演算で　＊、／、￥、ＭＯＤを使うことはできません。

(正)	Ｘ＄＝Ａ＄＋Ｂ＄	(誤)	Ｘ＄＝Ａ＄＊Ｂ＄
			Ｘ＄＝Ａ＄／Ｂ＄
			Ｘ＄＝Ａ＄￥Ｂ＄
			Ｘ＄＝Ａ＄　ＭＯＤ　Ｂ＄

文字列の比較

文字列は、数値の比較と同様に、関係演算子を用いて比較することができます。

半角文字には、キャラクタコードと呼ばれる番号が割り振られているので、それによって大小を比較します。キャラクタコードは、０～２５５の整数です。

文字列の比較は､比較する２つの文字列を先頭から順に１文字ずつ比べて行ないます｡すべてのキャラクタコードが互いに等しいとき、その２つの文字列は等しいと判断されます。

(例)　"ＨＥＬＬＯ"　＝　"ＨＥＬＬＯ"

比べて行く途中で、キャラクタコードの違うところが現われると、大きいコードの現われたほうが大きいと判断されます。

(例)　"ＭＩＬＫ"　＞　"ＭＩＬＤ"
　　　"ＳＨＯＰ"　＞　"ＳＨＩＰ"
　　　"ＴＥＬＬ"　＞　"ＴＥＬＥＰＨＯＮＥ"

比べて行く途中で、一方の文字列が終わってしまった場合は、その短い文字列のほうが小さいと判断されます。

(例)　"ＰＥＮＣＩＬ"　＞　"ＰＥＮ"

文字列の比較においては、空白も意味をもっています。

(例)　"ＭＹＳＴＥＲＹ"　＞　"ＭＹ　ＳＴＯＲＹ"

""（ヌルストリング）はどの文字列よりも小さい、最小の文字列として定義します。

(例)　"Ａ"　＞　""

文字列の比較によって、文字列のデータをアルファベット順に並び換えるなどのソーティング[1]ができます。

1）ソーティング（ｓｏｒｔｉｎｇ）　与えられたデータの集合の順序を、ある一定の規則に従って並び換えることです。ソーティングされたデータの例として、各種の辞書、電話帳、本の索引などがあります。

1.10　式

式とは、オペランド（定数、変数および関数）を演算子で結びつけたもの、あるいは単独の定数、変数および関数のことをいい、結果としてある値が得られるものをいいます。

演算子で結びつけられる定数、変数および関数は、すべて数値型か文字型のどちらかに一致していなければなりません。

式には、数式（算術式)、関係式、論理式、文字式の４つの種類があります。

1.10.1　数式

式のうち、２つのオペランドの間に算術演算子を書いたもので、算術演算を表現し結果として数値を得るものを数式（または算術式）といいます。単独の数値定数、数値変数、配列、および関数も数式とみなします。

例）４４０
　　＆ＨＢＣ３Ｆ
　　Ａ３
　　（Ｘ＋Ｙ）／２
　　ＳＩＮ（Ｘ）＾２＋ＣＯＳ（Ｘ）＾２

1.10.2　関係式

式のうち、２つのオペランドの間に関係演算子を書いたもので、両オペランドの値を比較するものを関係式といい、結果として真（－１）か偽（０）の値を得ます。オペランドは、数式、文字式のどちらでもかまいません。

（例）Ａ＞０
　　Ｘ＞＝Ｙ
　　Ｎ＝１００
　　Ａ＄＝"ＥＮＤ"

1.10.3　論理式

式のうち、定数、変数および関数を、関数演算子および論理演算子で結びつけたものを特に論理式といい、結果として真（－１）か偽（０）の値を得ます。

別名、条件式ともいいます。

（例）Ｘ＞０
　　Ｘ＞０　ＡＮＤ　Ｘ＜１０
　　Ｘ＜０　ＯＲ　Ｘ＞１０
　　Ａ－Ｂ＜１Ｅ－８　ＯＲ　Ｂ－Ａ＜１Ｅ－８
　　ＡＢＳ（Ａ－Ｂ）＜１Ｅ－８

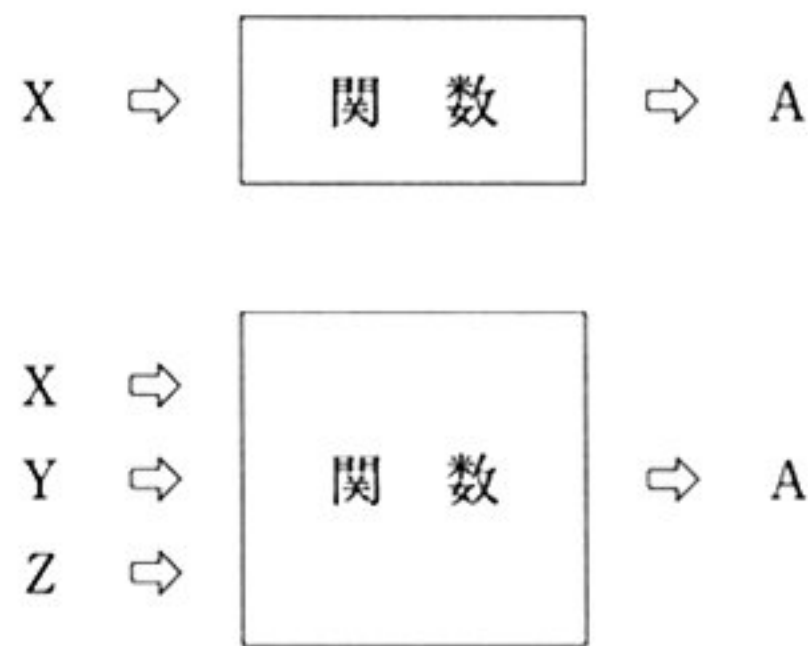
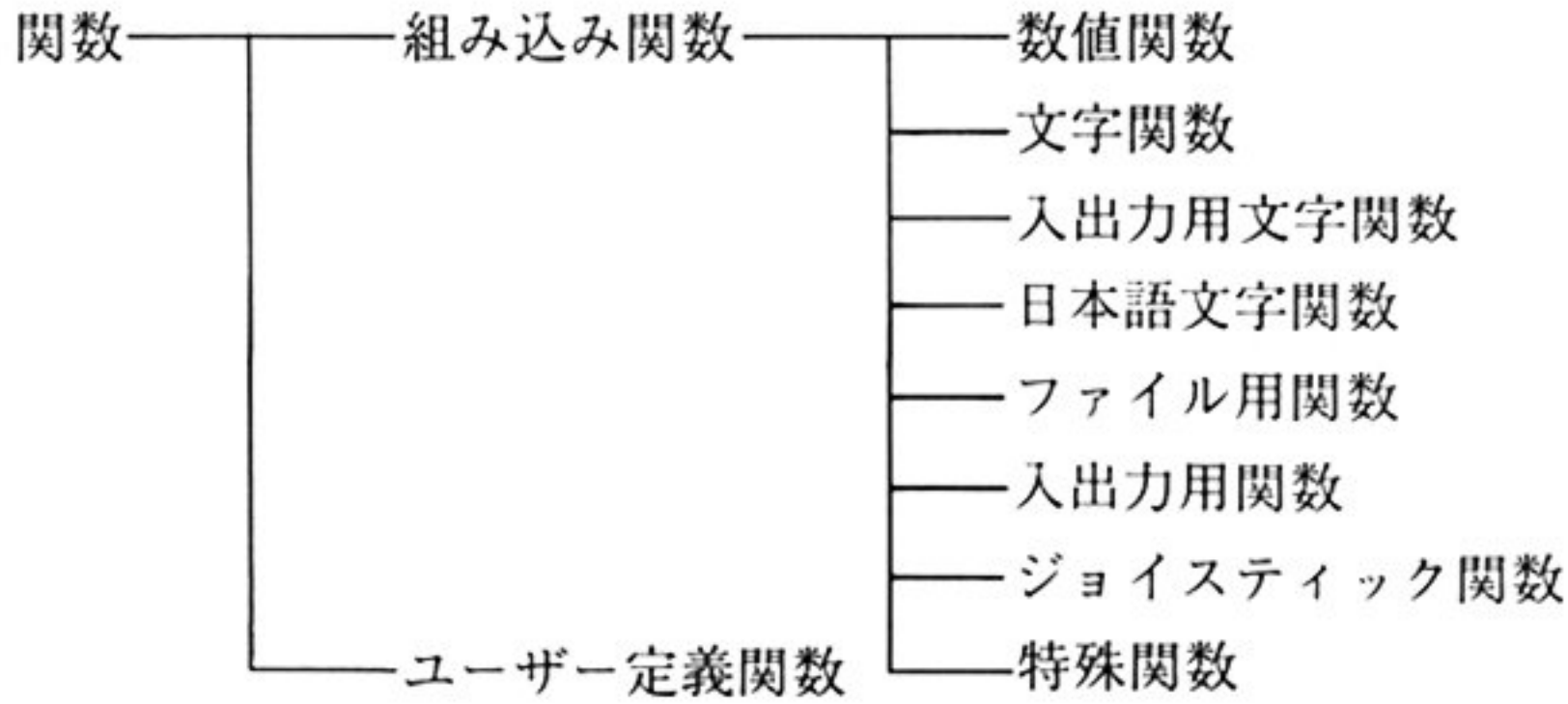
関係式と論理式は、各々および組み合わせて条件式として、ＩＦ－ＴＨＥＮ－ＥＬＳＥ文の中に書き、プログラムの分岐に用いることができます。(ＩＦ－ＴＨＥＮ－ＥＬＳＥ参照)

1.10.4　文字式

式のうち、単独の文字列、文字変数、関数、およびそれらを演算子（＋）で連結したもので、結果が文字列を表わすものを文字式といいます。

例）"ＨＥＬＬＯ"
　　"＆Ｈ"＋Ａ＄

1.11 関数とサブルーチン

関数（function）とは、与えた任意の値に対し、コンピュータが1対1あるいは多対1の対応で1つの値を返す機能です。

関数には、次のような種類があります。

サブルーチンは、それを呼び出すプログラムのサブプログラムで、プログラム中何度も出てくる処理をサブルーチンにすることにより、プログラムは判りやすくなります。

サブルーチンには、BASICのサブルーチンと機械語サブルーチンがあります。関数もサブルーチンの1種と考えることができます。

BASICのサブルーチンは、GOSUB文によって呼び出されます（2.2.15 GOSUB参照）

（例）

```
1110  GOSUB  10000……10000行から始まるサブルーチンを呼び出します。
......
......
10000  R=INT(RND*100)
10010  RETURN
```

機械語サブルーチンは、CALL文およびUSR関数によって呼び出されます。（2.2.26 CALL、3.8.8 USR関数、『ユーザーズマニュアル』の「機械語サブルーチンとモニタ」参照）

（例）

```
1500  CALL  &HE000……E000番地（16進数）から始まるサブルーチンを呼び出します。

1500  DEFUSR1=&HE000
1510  A=USR1(0)      ……E000番地（16進数）から始まるサブルーチンを呼び出します。
```

2 章

コマンド・ステートメント

コマンド・ステートメント・関数説明の形式と見方

ＢＡＳＩＣの各コマンド、ステートメント、関数およびシステム変数の各命令は、次のように、働き、文法、省略形、文例、説明、参照、サンプルプログラムの７つの項目に分けて記述してあります。

機　能　　**命令の働きを一口で書いたものです。これを読むことによって、命令のおおよその意味をつかむことができます。**

書　式　　命令の文法が書かれています。これによっておおよその命令の書きかたを知ることができます。

なお、ここで使っている記号や文字には次のような約束があります。

1. 英字の大文字はそのまま書きます。
2. 〔　〕の中は、ユーザーが任意に省略することができます。
3. かっこ（　）、カンマ（,）、コロン（:）、セミコロン（;）、ハイフン（−）、等号（＝）などの記号は、指定された位置に正しく入力します。
4. …………は、１行（２５５文字）の範囲で任意の回数繰り返すことができることを意味します。
5. 空白（間隔文字）のあるところは、空白をとることを意味しますが、この空白を無視して詰めてもよい場合もあります。
6. "　"の中は、文字列データであることを示しています。
7. x は数式、x $ は文字式、l、m、n は整数、a はアドレスを示します。
8. **fileは"デバイス名：〔パス名〕ファイル名"**の形式で書かれるファイルの指定を示します。
9. その他文法の中で使っている語句については、文法の記述のすぐ下の部分が説明のところで詳しく説明しています。

省 略 形　　その命令の短縮形を示します。省略形がない場合は、この部分はありません。

文　例　　簡単なステートメントの例を示しています。

解　説　　命令の使用方法や働き、注意事項を詳しく述べています。

参　照　　関連した命令や項目が示されています。

サンプル プログラム　　適当なサンプルプログラムを示し、その解説を行なっています。

その他、SHIFT ＋ BREAK は SHIFT キーを押しながら BREAK キーを押すという意味をもっています。

2.1 コマンド

2.1.1 CLEAR／LIMIT

機能

BASICが使用するメモリの最上位アドレスを設定します。
(⇔NEWON)

書式

〔1〕CLEAR a
〔2〕LIMIT a

. a：BASICが使用するメモリの最上位のアドレス＋1。&HF400以下。

省略形

〔1〕CLE.
〔2〕LIM.

文例

CLEAR &HC000
⇨BASICが使用するメモリの最上位のアドレスを&HBFFFに設定します。

解説

機械語プログラムとBASICプログラムを併用する場合、機械語プログラムはBASICプログラムより上方のメモリに記憶されます。このとき、BASICプログラムが機械語プログラムを侵食しないように、BASICプログラムの上方に防波堤をつくって、機械語プログラムの場所を確保しなければなりません。

CLEARまたはLIMITで指定するアドレスaは、防波堤の位置を示しa−1までをBASICプログラムの領域として使用することができます。

このコマンドを実行すると、防波堤の位置を定めると同時に、プログラムのネスティング[1)]が消去されます。しかし、変数、配列は消去されずに残ります。

このコマンドは、通常、プログラムの先頭部分に書きます。1度指定したアドレスは、再びCLEARまたはLIMITで指定するまで保持されます。

なお、このコマンドをサブルーチンの中に書くことはできません。

CLEARの後ろのアドレスaを省略すると、CLRと同じ働きをします。

1）ネスティング（nesting）自分自身と類似した構造を中に含む処理をネスティング、または入れ子の構造といいます。
FOR−NEXT、WHILE−WEND、REPEAT−UNTILおよびGOSUBにおいて、何重かのネスティングを形成することができます。

音訓辞書の入ったディスクBASIC（CZ−8FB02）を起動すると、自動的に、"Start up."プログラムが実行され、

CLEAR &HF000

が実行されます。音訓辞書をつかわず、1字変換機能だけを使うときは、

ＣＬＥＡＲ　＆ＨＦ４００

を実行してください。１Ｋバイト、メインメモリのフリーエリアが増えます。

このコマンドを実行すると、システム辞書のエリア・音訓辞書・ユーザー辞書のエリア（＆ＨＦ０００～＆ＨＦ３ＦＦ）をフリーエリアとして使用できるように自動的に日本語入力モードが１字変換に切り換わります。

ＣＬＥＡＲステートメントでＢＡＳＩＣの使用する最上位アドレスを各種辞書のエリアの下位に設定しているとき、日本語入力モードに入る場合は、辞書のエリアを誤って機械語プログラムで破壊しないよう注意して使ってください。

参　照

『ユーザーズマニュアル』の「機械語サブルーチンとモニタ」、
『ユーザーズマニュアル』付録の「メモリマップ」、
2.1.2 ＮＥＷＯＮ、2.2.3 ＣＬＥＡＲ／ＣＬＲ

2.1.2 NEWON

機　　能

ＢＡＳＩＣが使用するメモリの最下位アドレス（テキストの開始アドレス）を指定します。
(⇦⇨CLEAR／LIMIT)

書　　式

〔1〕 **NEWON〔n〕**
〔2〕 **NEWON〔a〕**

n：0～9までの整数（ＮＥＷＯＮとの間にスペースを入れないでください）
a：ＢＡＳＩＣが使用するメモリの最下位アドレス。すなわち、プログラムの開始アドレス。

文　　例

ＮＥＷＯＮ４
⇨**解説**に示された表のｎ＝４以降（４～９）に該当するコマンドやステートメントなどが削られフリーエリアが増えます。
ＮＥＷＯＮ　＆ＨＡ000
⇨ＢＡＳＩＣテキストの開始アドレスを＆ＨＡ000に設定し、現在のテキストおよび変数をすべて消去します。

解　　説

〔1〕ｎを指定するとＢＡＳＩＣ内のコマンドやステートメントなどが削られてフリーエリアが増加します。ディスクＢＡＳＩＣを起動した直後、

```
NEWON■
     ↑
     カーソル
```

と画面に表示されます。
0～9までの数値を入力するか、または数値を省略してリターンキー（⏎）を押してください。希望のレベルのＮＥＷＯＮが設定できます。

```
NEWON⏎
```

とキー入力すれば、すべてのコマンド、ステートメントが使用できます。
１度このコマンドを実行するとＢＡＳＩＣを再起動させないかぎり削られたコマンドやステートメントなどは実行できません。
実行しようとしても「Reserved feature」というエラーメッセージが表示されて実行できません。

n	
省　略	すべてのコマンド、ステートメント、関数など使用可
9	ＭＩＲＲＯＲ＄、ＫＡＮＪＩ＄、ＤＴＬ、ＲＡＮＤＯＭＩＺＥ、ＷＡＩＴ、ＫＥＹＬＩＳＴ、ＫＬＩＳＴ、ＫＢＵＦ、ＣＡＮＶＡＳ、ＬＡＹＥＲ、ＴＶＰＷ、ＣＨＡＮＮＥＬ、ＶＯＬ
8	ＶＥＲＩＦＹ、ＬＯＡＤ？、"ＣＡＳ："、ＣＭＴ、ＣＭＴ関数、ＲＥＷ、ＦＡＳＴ、ＥＪＥＣＴ、ＡＰＳＳ
7	"ＣＯＭ："、ＯＮ　ＣＯＭ　ＧＯＳＵＢ、ＣＯＭ　ＯＮ／ＯＦＦ／ＳＴＯＰ、ＰＯＳＩＴＩＯＮ、ＰＡＴＴＥＲＮ、ＣＩＲＣＬＥ＠、ＳＣＲＯＬＬ、 ＳＴＩＣＫ、ＳＴＲＩＧ、ＰＵＳＨ、ＰＯＰ、ＡＴＴＲ＄、ＲＵＮ"？．Ｓｙｓ"
6	ＣＲＴ、ＯＮ　ＫＥＹ　ＧＯＳＵＢ、ＫＥＹ　ＯＮ／ＯＦＦ／ＳＴＯＰ、ＯＮ　ＴＩＭＥ＄　ＧＯＳＵＢ、ＴＩＭＥ＄　ＯＮ／ＯＦＦ／ＳＴＯＰ
5	ＭＫＤＩＲ、ＣＨＤＩＲ、ＲＭＤＩＲ、ＨＤＯＦＦ、ＳＥＴ、ＮＡＭＥ、ＦＰＯＳ
4	ＭＯＵＳＥ、ＭＯＵＳＥ関数、ＨＣＯＰＹ
3	ＧＥＴ＠、ＰＵＴ＠、ＣＧＰＡＴ＄、ＤＥＶＩ＄、ＤＥＶＯ＄、ＬＰＯＵＴ、ＣＯＮＳＯＬＥ＃、ＣＯＰＹ
2	ＬＩＳＴ、ＬＬＩＳＴ、ＤＥＬＥＴＥ、ＲＥＮＵＭ、ＡＵＴＯ、ＥＤＩＴ、ＴＲＯＮ、ＴＲＯＦＦ、ＳＡＶＥ、ＳＥＡＲＣＨ、ＫＩＬＬ、ＣＯＮＴ、ＳＡＶＥＭ、 エラーメッセージ
1	ＰＬＡＹ、ＭＵＳＩＣ、ＴＥＭＰＯ、ＳＯＵＮＤ、ＣＢＬＡＣＫ ＰＬＡＹ＠、ＭＵＳＩＣ＠、ＳＯＵＮＤ＠
0	ＷＩＮＤＯＷ、ＬＩＮＥ、ＰＳＥＴ、ＰＲＥＳＥＴ、ＣＩＲＣＬＥ、ＰＯＬＹ、ＰＡＩＮＴ、ＰＡＩＮＴ＠、ＳＹＭＢＯＬ、ＰＯＩＮＴ、ＰＲＷ、ＣＬＩＣＫ、ＮＥＷＯＮｎ、ＰＡＬＥＴ、ＰＡＬＥＴ＠

〔2〕ａを指定するとＢＡＳＩＣプログラムの開始アドレスをａに定めます。このとき、現在のプログラムおよび変数をすべて消去します。

ａの値は、必ずＢＡＳＩＣインタープリタの上方のアドレスになければなりません。

ａを省略すると、ＢＡＳＩＣ起動時に設定されたアドレスが設定されます。

参　　照　『ユーザーズマニュアル』の「フリーエリアについて」、
2.1.1 ＣＬＥＡＲ／ＬＩＭＩＴ

2.1.3 RUN

2.1

機能

プログラムの実行を始めます。

書式

```
〔1〕RUN 〔n〕
〔2〕RUN file
```

n：実行を始めたい行番号。1～65534の整数。
file："デバイス名：〔パス名〕ファイル名"。
『ユーザーズマニュアル』の「ファイルについて」参照。
デバイス名：使用可能なデバイス名は次の通りです。
0：～3：、F0：～F3：、HD0：～HD3：、CAS：、
MEM0：～MEM1：、EMM0：～EMM9：、COM：

省略形

R.

文例

〔1〕RUN 100
⇨行番号100からプログラムを実行します。
〔2〕RUN"1：TEST"
⇨ディスクのドライブ1から"TEST"という名のプログラムを読み込んで、自動的に実行します。

解説

RUNは、プログラムを実行するためのコマンドです。RUNをダイレクトモードで入力すると、プログラムの先頭から実行が始まりますが、行番号nを指定すると、その行から実行を始めることができます。

SHIFT + BREAK キーを押すと、プログラムの実行を止めることができます。その後、実行を再開するときにRUNを使うと、そのときの変数の状態をすべて消去してしまうので、変数の状態を保存したまま実行を再開したいときは、GOTOかCONTステートメントを使ってください。

RUNの後ろにデバイス、パス名およびファイル名を指定すると、指定したファイル名のプログラムをロードし、実行を始めます。指定したプログラムがロードされる前に、オープンされているすべてのファイルはクローズされます。

また、システムファイル（ファイル名のエクステンションにSysのあるもの）も実行でき、特にシステムRUNと呼びます。

参照

2.2.14 GOTO、2.1.4 CONT、
『ユーザーズマニュアル』の「ファイルについて」

サンプル
プログラム

```
10 ' RUN (例)
20 PRINT"こんにちは。"
30 MUSIC"O4C"
40 PRINT"私はパソコンテレビです。"
50 MUSIC"O4E"
60 PRINT"どうぞよろしく。"
70 MUSIC"O4G"
```

```
RUN
こんにちは。
私はパソコンテレビです。
どうぞよろしく。
Ok
RUN40
私はパソコンテレビです。
どうぞよろしく。
Ok
RUN60
どうぞよろしく。
Ok
```

ＲＵＮ〔ｎ〕の例です。文字と表示も音が鳴ります。

2.1.4 CONT

機　能　**中断したプログラムの実行を再開します。(⇦⇨STOP)**

書　式

```
CONT
```

省略形　C.

文　例　CONT

⇨ＳＴＯＰや BREAK キーで中断したプログラムの実行を再開します。

解　説

SHIFT ＋ BREAK キーを押すか、ＳＴＯＰステートメントを実行すると、プログラムを止めることができることがあります。この後、ＣＯＮＴコマンドをダイレクトモードで入力すると、変数の状態を破壊しないで、止まった次のステートメントから実行を再開することができます。

しかし、ＥＮＤステートメントの実行後、エラーの起こった後、ＣＬＥＡＲステートメントの実行直後、あるいはプログラムを止めてから書き換えを行なった後は、ＣＯＮＴコマンドで実行を再開することはできず、「Can't continue」のエラーが出ます。

実行が再開できるか否かは、画面上の「Ok」の後に「.」があるかないかで判断します。「.」があるときは再開できますが、ないときは再開できません。

ＣＯＮＴコマンドは、デバックのためにＳＴＯＰステートメントといっしょに使います。プログラム中にＳＴＯＰを挿入して実行し、止まったときに、ＰＲＩＮＴで変数の値を調べて、プログラムが正しく作動しているかチェックすることができます。チェックが終わったら、ＣＯＮＴを使って実行を再開します。

参　照　2.2.6 STOP

2.1

サンプルプログラム

```
10 ' CONT (例)
20 FOR I=0 TO 20
30 PRINT "I=";I;TAB(8);SQR(I)
40 NEXT
RUN
I= 0      0
I= 1      1
I= 2      1.4142136
I= 3      1.7320508
I= 4      2
I= 5      2.236068
I= 6      2.4494897
I= 7      2.6457513
I= 8      2.8284271
Break in 30
Ok.
? I,SQR(I),I^2
 8            2.8284271            64
Ok.
CONT
I= 9      3
I= 10     3.1622777
I= 11     3.3166248
I= 12     3.4641016
I= 13     3.6055513
I= 14     3.7416574
I= 15     3.8729833
I= 16     4
I= 17     4.1231056
I= 18     4.2426407
I= 19     4.3588989
I= 20     4.4721359
Ok
```

2.1.5 TRON／TROFF

機　　能　プログラムの動きを追跡します。

書　　式

〔1〕TRON
〔2〕TROFF

省 略 形

〔1〕T.
〔2〕TROF.

文　　例

〔1〕TRON
⇨TRONを実行後RUNコマンドでプログラムを実行すると、実行した行の行番号を〔　〕で囲んで、実行順に画面に表示します。

〔2〕TROFF
⇨プログラムの実行の追跡を解除します。

解　　説

このコマンドは、プログラムのデバッグのために使います。
TRONコマンドを入力してプログラムを実行すると、実行した行番号を大かっこ〔　〕で囲んで画面に表示します。表示された行番号を見れば、確実に、実行されたプログラム行を知ることができるので、予測しないプログラムの動きをみつけて、論理的なミスを発見することができます。

プログラムの実行を一時的に停止するには、BREAKキーを押します。プログラムの実行を完全に停止して、コマンドの入力待ちの状態に戻すには、SHIFT+BREAKキーを押します。

プログラムの実行を解除するには、TROFFコマンドを入力します。

TRONとTROFFはプログラム中に挿入できるので、プログラム中のある区間だけを追跡することができます。

サンプルプログラム

```
10 ' TRON/TROFF <例>
20 FOR I=1 TO 10
30 R=INT(RND*2)+1
40 IF R<2 THEN 60
50 PRINT"Hi!"
60 NEXT
TRON
Ok
RUN
[10][20][30][40][50]Hi!
[60][30][40][50]Hi!
[60][30][40][50]Hi!
[60][30][40][50]Hi!
[60][30][40][60][30][40][50]Hi!
[60][30][40][60][30][40][50]Hi!
[60][30][40][60][30][40][50]Hi!
[60]
Ok
TROFF
Ok
RUN
Hi!
Hi!
Hi!
Hi!
Hi!
Hi!
Hi!
Ok
```

2.1.6 MON

機　能　**機械語モニタを起動します。**

書　式

```
MON
```

文　例

ＭＯＮ

⇨機械語モニタが起動し、「*」が画面に表示されモニタ用コマンドの入力待ちの状態になります。

解　説

ＭＯＮは、機械語モニタを起動するためのコマンドです。

機械語モニタの詳しい使用方法については、『ユーザーズマニュアル』の「機械語サブルーチンとモニタ」を参照してください。

　Ｍキー押した状態で電源入力またはＩＰＬリセットを実行した場合、機械語モニタを起動できます。

参　照

『ユーザーズマニュアル』の「機械語サブルーチンとモニタ」

2.1.7 BOOT

機能 IPLを起動します。

書式

```
BOOT
```

省略形 BO.

文例 BOOT

⇨IPLを起動するか否か聞いてくるので起動したければ[Y]キーを、起動したくなければ[N]キーを押します。

解説 BOOT[↵]と打ち込むと、

```
BOOT↵
Are you sure? (y or n) ■←カーソル
```

と表示され、キーの入力を待っているので、IPL(Initial Program Loader)を起動したければ、[Y]キーを、起動したくなければ、[N]キーを押してください。

[Y]キーを押すと、電源投入直後の状態になり、現在のプログラムがすべて消去されます。

プログラム中にBOOTステートメントを入れると、

Are you sure? (y or n) ■

と聞いて来ないで、直接IPLを起動してしまうので、注意してください。

2.1.8　ＡＳＫ

機　能　**内蔵の予約タイマーを呼びだします。**

書　式

```
ASK
```

文　例

ＡＳＫ
⇨ＴＶ　Ｔｉｍｅｒ　ｃｏｎｔｒｏｌ（テレビタイマーコントロール）になります。

解　説

ＡＳＫ⏎を入力すると、内蔵の予約タイマーを呼び出すことができます。
予約タイマーについて、詳しくは『ユーザーズマニュアル』の「テレビコントロール」をお読みください。

参　照

『ユーザーズマニュアル』の「テレビコントロール」

2.1.9 AUTO

機　能　自動的に行番号を発生させます。

書　式

```
〔1〕AUTO〔m〕〔,〔n〕〕
〔2〕AUTO＊〔m〕〔,〔n〕〕
```

m：発生させる最初の行番号。1～65534の整数。
n：行番号の間隔。1～65534の整数。
＊：行番号の後ろにREM（'）をつける指定。

省略形　A.
A.＊

文　例　AUTO　1000
⇨1000番から行番号を発生し⏎キーを押すたびに1010、1020……と10ずつ増加した値を発生するようになります。
AUTO＊
⇨1000番からREMの省略形「'」のついた行番号を発生します。⏎キーを押すたびに1010'、1020'……と10番ずつ増加した値を発生するようになります。

解　説

BASICのプログラムを入力する際には、必ず各行の初めに行番号をつけなければなりません。しかし、行番号をいちいちタイプするのは面倒なので、AUTOコマンドを使って、自動的に行番号を発生させるようにします。

mとnの指定のしかたによって、次のような使い方ができます。

(1)　AUTO⏎　……10番から行番号を発生し、⏎キーを押すたびに10ずつ増加した値を発生し続けます。
(2)　AUTO　m⏎　……m番から行番号を発生し、⏎キーを押すたびに10ずつ増加した値が発生し続けます。
(3)　AUTO　m，n⏎　……m番から行番号を発生し、⏎キーを押すたびにnずつ増加した値が発生し続けます。
(4)　AUTO　，n⏎　……行管理ポインタの指す行番号から発生し、⏎キーを押すたびにnずつ増加した値が発生し続けます。
(5)　AUTO　．⏎　……行管理ポインタの指す行番号から発生し、⏎キーを押すたびに10ずつ増加した値が発生し続けます。

上の各AUTOにアスタリスク（＊）を指定してAUTO＊とすると、アポストロフィ（'）＝REMがついた行番号が自動的に表示されます。（⇨『ユーザーズマニュアル』の「日本語処理」参照）

（例1）AUTO＊⏎……1000番から「'」のついた行番号を発生し⏎キー

を押すたびに１０番ずつ増加した値が発生し続けます。

（例２） ＡＵＴＯ＊　１００⏎…１００番から「'」のついた行番号を発生し、⏎キーを押すたびに１０ずつ増加した値が発生し続けます。
１００'
１１０'

ＡＵＴＯの発生する行番号で、すでにプログラム中に存在する番号があると、そのプログラム行が表示されます。このとき新たな入力があると、その行が書き換わりますが、入力しないでそのまま⏎キーを押せば書き換わらず、ＡＵＴＯは次の行番号を発生します。

行番号の発生を解除するには、SHIFT＋BREAKキーを押します。解除したときのプログラム行はメモリに保存されません。

参　照

『ユーザーズマニュアル』の「日本語処理」

サンプルプログラム

```
NEW
Ok
LIST
Ok
AUTO
10 ' AUTO (例1)
20 PRINT"AUTO (例1)"
30 END
40
NEW
Ok
LIST
Ok
AUTO 100
100 ' AUTO (例1)
110 PRINT"AUTO (例1)"
120 END
130
NEW
Ok
LIST
Ok
AUTO 100,20
100 ' AUTO (例1)
120 PRINT"AUTO (例1)"
140 END
160
NEW
Ok
LIST
Ok
AUTO ,50
10 ' AUTO (例1)
60 PRINT"AUTO (例1)"
110 END
160
NEW
Ok
LIST
Ok


NEW
Ok
LIST
Ok
AUTO*
1000 ' AUTO (例2)
1010 '   平家物語
1020 '   祇園精舎の鐘の声、諸行無常の響きあり。
1030 '   沙羅双樹の花の色、盛者必衰のことわりをあらはす。
1040 '
```

2.1.10　ＬＩＳＴ

機　能　プログラムを画面あるいはその他の指定されたデバイス上にリストします。

書　式

```
〔1〕ＬＩＳＴ　〔〔m〕〔－〔n〕〕〕
　　 ＬＩＳＴ　〔〔m〕〔,〔n〕〕〕
〔2〕ＬＩＳＴ＊　〔〔m〕〔－〔n〕〕〕
　　 ＬＩＳＴ＊　〔〔m〕〔,〔n〕〕〕
〔3〕ＬＩＳＴ　〔file〕〔,〔m〕〔－〔n〕〕〕
　　 ＬＩＳＴ　〔file〕〔,〔m〕〔,〔n〕〕〕
〔4〕ＬＩＳＴ＊　〔file〕〔,〔m〕〔－〔n〕〕〕
　　 ＬＩＳＴ＊　〔file〕〔,〔m〕〔,〔n〕〕〕
〔5〕ＬＩＳＴ＠　〔〔m〕〔－〔n〕〕〕
　　 ＬＩＳＴ＠　〔〔m〕〔,〔n〕〕〕
```

m：リストする最初の行番号。１～６５５３４の整数。

n：リストする最後の行番号。１～６５５３４の整数。

＊：リストするとき、注釈文のみ表示する指定。

file：『ユーザーズマニュアル』の「ファイルについて」参照。

使用可能なデバイス名は次の通りです。

０：～３：、Ｆ０：～Ｆ３：、ＨＤ０：～ＨＤ３：、ＭＥＭ０：～ＭＥＭ１：、ＥＭＭ０：～ＥＭＭ９：、ＣＡＳ：、ＬＰＴ：、ＳＣＲ：、ＣＲＴ：、ＣＯＭ：

省略形

Ｌ．

Ｌ．＊

Ｌ．＠

文　例

ＬＩＳＴ　１００－２００

⇨プログラムの１００番から２００番をリストします。

ＬＩＳＴ＊

⇨プログラムのうちアポストロフィ（'）で始まる注釈文のみを表示します。このとき行番号と「'」は表示されず文のみが表示されます。

ＬＩＳＴ＠１００－２００

⇨プログラムの１００番から２００番をリストします。これを実行後ＬＩＳＴを実行するとＬＩＳＴ１００－２００と同じ実行になります。

解　説

ＬＩＳＴは、プログラム画面あるいは他の指定されたデバイス上にリストするコマンドですが、ほとんどの場合、プログラムを画面にリストしてその内容をみるために使います。

行番号を指定することによって、プログラムの任意の部分をみることができます。

（1）ＬＩＳＴ　ｍ⏎　　……ｍ番のみリストします。
（2）ＬＩＳＴ　ｍ－⏎　……ｍ番以降、プログラムの最後までをリストします。
（3）ＬＩＳＴ　ｍ－ｎ⏎　……ｍ番からｎ番までをリストします。
（4）ＬＩＳＴ　－ｎ⏎　……プログラムの最初からｎ番までをリストします。
（5）ＬＩＳＴ⏎　　……プログラム全体をリストします。

指定した行番号がプログラムになかった場合は、次のようにリストされます。

（1）のときは、何も表示しないで次のコマンドの入力待ちになります。

（2）のときは、ｍより大きくて１番近い番号からリストを始めます。ｍ番以降がなければ、何も表示しないで次の入力待ちになります。

（4）のときは、プログラムの最初からｎより小さくて最も近い番号までリストします。ｎ番以前がなければ、何も表示しないで次の入力待ちになります。

（3）のときは、はじめのｍについては（2）と同じ要領で、後のｎについては（4）と同じ要領でリストします。

「.」は、ＢＡＳＩＣ内部の行管理ポインタの役割をもっているので、次のような使い方ができます。

ＬＩＳＴ．⏎（または省略形のＬ．．⏎）
……現在行管理ポインタが指している行番号のみをリストします。

ＬＩＳＴ．－⏎（または省略形のＬ．．－⏎）
……現在、行管理ポインタが指している行番号以降をリストします。

ｆｉｌｅによってデバイスとファイル名を指定すると、指定されたデバイスにアスキー形式でセーブされます。

（例1）ＬＩＳＴ　"０：ＴＥＳＴ"
……ディスクドライブ０にＴＥＳＴというファイル名でプログラムがアスキー形式でセーブされます。

（例2）ＬＩＳＴ　"１：ＴＥＳＴ"，１００－９００
……ディスクドライブ１にＴＥＳＴというファイル名で、プログラムの１００～９００番がアスキー形式でセーブされます。

ディスクドライブにリストを行なうと、プログラムの指定した部分がアスキー形式でセーブされます。こうしてリストしたファイルは、後でＭＥＲＧＥに使うことができます。

ｆｉｌｅに"ＳＣＲ："や"ＣＲＴ："を指定すると、画面を指定したことになり、ｆｉｌｅを指定しない〔1〕の書式と同じ意味になります。また、"ＬＰＴ："を指定するのは後出のＬＬＩＳＴを使うのと同じ意味になります。

さらに、ＬＩＳＴの後ろに＊を指定してＬＩＳＴ＊として使用すると、リスト表示を行なう際にアポストロフィ（'）で始まる注釈文のみを表示することができます。このとき、注釈文の内容のみを表示し、「'」および行番号は表示しません。

なお、画面上にリストを出している最中、BREAKキーを押すと一時的に止めることができます。リストを再開するときは、任意のキーを押してください。リストを完全に止めて、コマンド入力待ちの状態に戻すには、SHIFT＋

BREAK キーを押してください。

ＬＩＳＴ@はｍとｎを指定するとｍ行からｎ行までリストし、それ以後ＬＩＳＴ⏎を実行してもその指定された範囲のみをリストします。

つまり、ＬＩＳＴ@は必要のない行の表示をさけ、必要な行だけ表示するステートメントといえます。

また、ＬＩＳＴ@⏎で、m，ｎを省略すると、プログラム全体をリストし、その後、ＬＩＳＴ⏎としてもプログラム全体をリストします。

なお、ＬＩＳＴ@m－ｎ⏎を実行した後、ＳＡＶＥ〔file〕，Ａ⏎を実行してアスキーセーブしても行番号ｍ－ｎ以外の行はセーブされません。

参　照

2.1.11 ＬＩＳＴ、『ユーザーズマニュアル』の「ファイルについて」、「日本語処理」

サンプルプログラム

（例１）

```
NEW
Ok
LIST
Ok
10 ' LIST (例1)
20 PRINT"LIST (例1)"
30 INPUT"X=";X
40 PRINT"SQR(X)=";SQR(X)
50 END
LIST
10 ' LIST (例1)
20 PRINT"LIST (例1)"
30 INPUT"X=";X
40 PRINT"SQR(X)=";SQR(X)
50 END
Ok
LIST 30
30 INPUT"X=";X
Ok
LIST 40-
40 PRINT"SQR(X)=";SQR(X)
50 END
Ok
LIST 20-40
20 PRINT"LIST (例1)"
30 INPUT"X=";X
40 PRINT"SQR(X)=";SQR(X)
Ok
LIST -20
10 ' LIST (例1)
20 PRINT"LIST (例1)"
Ok
```

行番号１０００～１０３０をＫｅｙより入力してください。ＬＩＳＴとＬＩＳＴ＊は表示が違います。

（例２）

```
NEW
Ok
LIST
Ok
1000 ' LIST (例2)
1010 '   平家物語
1020 '   祇園精舎の鐘の声、諸行無常の響きあり。
1030 '   沙羅双樹の花の色、盛者必衰のことわりをあらはす。
LIST
1000 ' LIST (例2)
1010 '   平家物語
1020 '   祇園精舎の鐘の声、諸行無常の響きあり。
1030 '   沙羅双樹の花の色、盛者必衰のことわりをあらはす。
Ok
LIST*
 LIST (例2)
   平家物語
   祇園精舎の鐘の声、諸行無常の響きあり。
   沙羅双樹の花の色、盛者必衰のことわりをあらはす。
Ok
```

2.1.11 LLIST

機　能

プログラムをプリンタにリストします。

書　式

```
〔1〕LLIST  〔〔m〕〔－〔n〕〕〕
     LLIST  〔〔m〕〔,〔n〕〕〕
〔2〕LLIST＊ 〔〔m〕〔－〔n〕〕〕
     LLIST＊ 〔〔m〕〔,〔n〕〕〕
```

m：リストする最初の行番号。1～65534の整数。
n：リストする最後の行番号。1～65534の整数。
＊：リストするとき、注釈文のみを表示する指定。

省略形

〔1〕LL.　〔2〕LL.＊

文　例

LLIST　100－500
⇨行番号の100～500までのプログラムをテキストをプリンタに打ち出します。

解　説

ＬＬＩＳＴは、プログラムをプリンタにリストするコマンドです。使用方法は、ＬＩＳＴの場合と同じです。
このコマンドは、ＬＩＳＴコマンドでfileに"ＬＰＴ："を指定して使うのを同じ働きをします。
ＬＬＩＳＴの後ろに＊を指定してＬＬＩＳＴ＊として使用すると、リスト表示をする際アポストロフィ（'）で始まる注釈文だけを表示することができます。
このとき、注釈文の内容のみ表示し、（'）および行番号は表示しません。

参　照

2.1.10 ＬＩＳＴ

サンプルプログラム

```
10 ' LLIST (例)
20 '  平家物語
30 '  祇園精舎の鐘の声、諸行無常の響きあり。
40 '  沙羅双樹の花の色、盛者必衰のことわりをあらわす。

LLIST
Ok
```

```
10 ' LLIST (例)
20 '  平家物語
30 '  祇園精舎の鐘の声、諸行無常の響きあり。
40 '  沙羅双樹の花の色、盛者必衰のことわりをあらわす。
```

```
10 ' LLIST (例)
20 '  平家物語
30 '  祇園精舎の鐘の声、諸行無常の響きあり。
40 '  沙羅双樹の花の色、盛者必衰のことわりをあらわす。

LLIST*
Ok
```

```
LLIST (例)
 平家物語
 祇園精舎の鐘の声、諸行無常の響きあり。
 沙羅双樹の花の色、盛者必衰のことわりをあらわす。
```

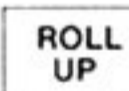
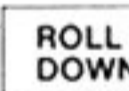
2.1.12 EDIT

機　能　**指定した行を表示し、カーソルをそこに移動します。**

書　式

```
EDIT 〔n〕
EDIT 〔.〕
```

n：行番号。1～65534の整数。

省略形　E.

文　例　EDIT　190
⇨行番号190がリストされ、その行が修正可能となります。

解　説　テキスト画面の左上隅に指定された行を表示し、nの右側にカーソルを移動して、入力待ちになります。このコマンドは、エラーの発生したプログラム行の修正を能率的に行なうために使います。エラーが発生すると、BASIC内部の行管理ポインタが「.」に入るので、

EDIT. ⏎（または省略形のE.. ⏎）

と入力すると、エラーの発生行がリストされ、直ちに修正の入力をすることができます。

EDITの実行中は、ROLL UP／ROLL DOWN キーにより指定された行の上下からテキスト全体にわたって画面に表示して、編集することができます。

EDIT状態はSHIFT＋BREAKキーを押すと解除されます。

LIST. ⏎（または省略形のL.. ⏎）

としても、同様にエラーの発生行をリストすることができますがこの場合はROLL UP／ROLL DOWN キーにより、その前後の行の表示することはできません。

参　照　2.1.10 LIST

2.1.13 RENUM

機　能　プログラムの行番号をつけなおします。

書　式

```
RENUM 〔〔l〕,〔m〕,〔n〕〕
```

l：これからつける新しい行番号。1～65534の整数。省略すると10。
m：行番号をつけかえたい現在のプログラムの行番号。1～65534の整数。省略すると、現在のプログラムの先頭の行番号。
n：行間の増分。1～65534の整数。省略すると10。

省略形　REN.

文　例　RENUM 1000

⇨プログラムの先頭を1000番にし、以後1010、1020、……と10番間隔で行番号をつけなおします。

解　説

プログラムを書いているとき、行番号が不揃いになったり、行間隔が狭くなって新しいプログラム行を挿入するのに困ることがあります。このようなとき、RENUMコマンドを実行すると、m番以降の行番号をlで始まる行番号につけなおしてくれます。新しい行番号は増分がnの等間隔になります。

RENUMを実行したとき、GOTO、GOSUB、IF－THEN－ELSEの各ステートメントのジャンプ先を示す行番号も自動的に変わります。もし、ジャンプ先を示す行番号がプログラム中になかった場合は65535になります。

l, m, nの3つのパラメータの指定の仕方によって、次のような使い方ができます。

（1）RENUM⏎　……プログラムの先頭を10番にし、間隔を10にします。
（2）RENUM l⏎　……プログラムの先頭をlにし、間隔を10にします。
（3）RENUM l, m⏎　……mから始まる行をlにし、間隔を10にします。
（4）RENUM l,, n⏎　……プログラムの先頭をlにし、間隔をnにします。
（5）RENUM , m⏎　……現在のプログラムを先頭よりmが小さいときのみ、プログラムの先頭を10にし、間隔を10にします。
（6）RENUM , m, n⏎　……現在のプログラムを先頭よりmが小さいときのみ、プログラムの先頭を10にし、間隔をnにします。
（7）RENUM ,, n⏎　……プログラムの先頭を10番にし、間隔をnにします。

(注) 現在あるプログラムの途中の行番号m1から指定する場合、新しい行番号lはm1の1つ前の行番号mより大きくなければなりません。(l>m)

サンプルプログラム

```
5 ' RENUM (例)
20 FOR I=0 TO 20
21 PRINT I;SQR(I)
53 NEXT
RENUM
Ok
LIST
10 ' RENUM (例)
20 FOR I=0 TO 20
30 PRINT I;SQR(I)
40 NEXT
Ok
RENUM 100
Ok
LIST
100 ' RENUM (例)
110 FOR I=0 TO 20
120 PRINT I;SQR(I)
130 NEXT
Ok
RENUM 200,110
Ok
LIST
100 ' RENUM (例)
200 FOR I=0 TO 20
210 PRINT I;SQR(I)
220 NEXT
Ok
RENUM 20,,20
Ok
LIST
20 ' RENUM (例)
40 FOR I=0 TO 20
60 PRINT I;SQR(I)
80 NEXT
Ok
```

2.1.14 SEARCH

機　能

探したい文字列を含むプログラム行をリストします。

書　式

```
SEARCH x$
```

x＄："文字列"。ただし２５５文字以内の文字列。

省略形

ＳＥ.

文　例

ＳＥＡＲＣＨ"Ａ＝"

⇨プログラム中から文字列Ａ＝を含んだ行を探し出し、画面にそのすべてをリストします。

解　説

ＳＥＡＲＣＨコマンドは、ｘ＄で指定された文字列を含むプログラム行を探し出し、画面上にリストします。

このコマンドは、デバッグの際に使うとたいへん便利なものです。

たとえば、ある定数、変数、配列、ラベルなどがどこにあり、どのような使われ方をしているのか調べたいときに使います。

ｘ＄の文字列中に引用符（"）を含めることはできません。

サンプルプログラム

```
10 ' SEARCH <例>
20 INPUT A,B,X
30 Y=A*X+B
40 PRINT A,B,X,Y
50 END
SEARCH"A"
10 ' SEARCH <例>
20 INPUT A,B,X
30 Y=A*X+B
40 PRINT A,B,X,Y
Ok
SEARCH"A,"
20 INPUT A,B,X
40 PRINT A,B,X,Y
Ok
```

2.1.15 DELETE

機　能　プログラムを部分的に消します。

書　式

```
〔1〕DELETE 〔m〕,〔n〕
〔2〕DELETE 〔m〕－〔n〕
```

m ：抹消を開始する行番号。1～65534の整数。
n ：抹消を終了する行番号。1～65534の整数。

省略形　D.

文　例　DELETE　100，200
⇨プログラムの行番号100から行番号200までを消します。

解　説　プログラムを作成していて削りたい行が出てきたとき、その行数が1、2行の場合は、その行の行番号を入れて⏎キーを押せば消すことができます。たとえば100番の行を消すときは100⏎を入力します。しかし、多くの行をまとめて削りたいときにこの方法では時間がかかりますので、DELETEコマンドを使います　DELETEを実行すると、m行からn行のすべての行を消すことができます。
mとnの指定の仕方によって、次のような使い方ができます。

（1）DELETE　m⏎　……m番のみ消します。
（2）DELETE　m－⏎　……m番以降を消します。
（3）DELETE　m－n ⏎　……m番からn番までを消します。
（4）DELETE　－n⏎　……先頭行からn番までを消します。
（5）DELETE　⏎　……行管理ポインタが指している1行のみ消します。

参　照　2.1.16 NEW

サンプル
プログラム


```
LIST
10 ' DELETE 〈例〉
20 INPUT"A,B,X=";A,B,X
30 INPUT"A,B,X=";A,B,X
40 Y=A*X+B
50 Y=A*X+B
60 Y=A*X+B
70 PRINT"A*X+B=";Y
80 PRINT"A*X+B=";Y
90 PRINT"A*X+B=";Y
100 END
Ok
DELETE 30,50     30行から50行まで消します。
Ok
LIST
10 ' DELETE 〈例〉
20 INPUT"A,B,X=";A,B,X
60 Y=A*X+B
70 PRINT"A*X+B=";Y
80 PRINT"A*X+B=";Y
90 PRINT"A*X+B=";Y
100 END
Ok
DELETE 80,       80行以降を消します。
Ok
LIST
10 ' DELETE 〈例〉
20 INPUT"A,B,X=";A,B,X
60 Y=A*X+B
70 PRINT"A*X+B=";Y
Ok
```

2.1.16 NEW

機　能

プログラムを消します。

書　式

```
NEW
```

解　説

新しいプログラムを入力するときは、現在あるプログラムを消してから行ないます。プログラムを消すには、NEW⏎と入力します。NEWは、プログラムを消すだけでなく、すべての変数を消し、すべてのファイルをクローズします。

なお、誤ってプログラムを消してしまうことのないように、NEWには省略形がありません。

参　照

2.1.15 DELETE

サンプルプログラム

```
10 ' NEW (例)
20 PRINT"テスト"
30 MUSIC"O4CEG"
LIST
10 ' NEW (例)
20 PRINT"テスト"
30 MUSIC"O4CEG"
Ok
RUN
テスト
Ok
NEW
Ok
RUN
Ok
LIST
Ok
```

2.1.17 DEVICE

機能 デバイス名の指定を省略したときに用いられるデバイス名およびディスクのタイプを決めます。

書式

```
DEVICE "デバイス名:〔n〕"
```

デバイス名:『ユーザーズマニュアル』の「デバイス名」参照。
n:フロッピーディスクのタイプ

0:~3に対して	0……320 KB	2D
	1……640 KB	2DD
	2……1MB	2HD
	3……1MB	＊2HD
F0:~F3に対して	0……1MB	2D−256
	1……1MB	＊2D−256
	2……1MB	＊1S−128

(詳しくは『ユーザーズマニュアル』の「ディスクの使い方」参照)

省略形 DEV.

文例 DEVICE "MEM0:"
⇨デバイス名省略時のデバイスをグラフィックメモリ0とします。

解説 ファイル入出力コマンド(FILES、LFILES、SAVE、SAVEM、NAME、COPY、SET、LOAD、LOADM、VERIFY、LOAD?、CHAIN、MERGE、KILL)、ファイル処理ステートメント(OPEN、DEVI$、DEVO$)、およびLIST、RUNのそれぞれにおいて、デバイス名を省略すると、DEVICEコマンドで指定されたデバイス名の示すデバイスが使用されます。nを指定するとディスクタイプの設定となりデバイス名は変わりません。

ファイルに関するコマンドやステートメントを使うプログラムにおいて、DEVICEで前もって固定して、デバイス名を何度も指定する手間を省きます。

DEVICEの実行後でも、他のデバイス名を指定すれば、そのデバイスを使うことができます。ディスクBASICを起動した時点では、BASICのフロッピーディスクの入っているドライブ番号が初期デバイスとして指定されます。

また、カセットBASICやROMのBASICが起動した時点では、

DEVICE "CAS:"

が実行されて、デバイス名としてカセットが指定された状態になっています。

デバイス名	指定されるデバイス
CRT:	画面
SCR:	画面
KEY:	キーボード
LPT:	プリンタ
CAS:	カセット
Ø:	ディスクØ ┐
1:	ディスク1 ├3or5インチ版
2:	ディスク2 │
3:	ディスク3 ┘
FØ:	ディスクØ ┐
F1:	ディスク1 ├8インチ版
F2:	ディスク2 │
F3:	ディスク3 ┘
HDØ:	ハードディスクØ
HD1:	ハードディスク1
HD2:	ハードディスク2
HD3:	ハードディスク3
EMMØ:	外部メモリØ
|	|
|	|
EMM9:	外部メモリ9
MEMØ:	グラフィックメモリØ
MEM1:	グラフィックメモリ1
COM:	RS－232Cポート

参照　『ユーザーズマニュアル』の「ファイルについて」

2.1.18 FILES

機　能　　指定したディレクトリに存在するファイル名をリストします。

書　式

```
FILES 〔"デバイス名：〔パス名〕"〕
```

デバイス名：『ユーザーズマニュアル』の「ファイルについて」参照。
　使用可能なデバイス名は次の通りです。
　　0：～3：、F0：～F3：、HD0：～HD3：、
　　MEM0：～MEM1、EMM0：～EMM9：、CAS：
　　CAS：のときはパス名を指定できません。
パス名：『ユーザーズマニュアル』の「ファイルについて」参照。

省略形　　FIL.

文　例　　FILES "1：営業部"
⇨3または5インチフロッピーディスク1の「営業部」というディレクトリに存在するファイルの一覧表を画面に表示します。

解　説

デバイス名で指定したデバイス内の指定したディレクトリに存在するファイル名を画面にすべてリストします。

ディスクに対してFILESを実行すると、最初に

```
nn Clusters free
Path name "デバイス名：パス名"
```

と、残りのクラスタ数と指定したディレクトリまでのパス名が示され、続いてファイルおよびディレクトリのリストが表示されます。

デバイス名の指定を省略すると、DEVICEコマンドで指定されたデバイスが指定されます。

パス名を省略すると、現在のカレントディレクトリに存在するファイル名がリストされます。

SHIFT + BREAK キーを押すと、リストを止めることができます。

なお、次のデバイスに対してはFILES（LFILESも同様）を使用することができません。

　画面（CRT：、SCR：）、キーボード（KEY：）、
　プリンタ（LPT：）RS-232Cポート（COM：）

〈注　意〉　カセットに対してFILESコマンドを実行すると、デバイス名として"CAS0："が表示されますが、"CAS："と同様です。

参　照　　2.1.19 LFILES、『ユーザーズマニュアル』の「ファイルについて」

サンプル
プログラム

```
OPTION SCREEN 3
Ok
FILES"MEM1:"
128 Clusters free
Path name "MEM1:/"
Ok
10 ' FILES (例)
20 PRINT"私はパソコンテレビです。"
30 END
SAVE"MEM1:TEST"
Ok
FILES"MEM1:"
127 Clusters free
Path name "MEM1:/"
Bas        "MEM1:TEST         .   "
'84/09/15 SAT 08:55
Ok
```

2.1.19 LFILES

機　能

指定したデバイスのカレントディレクトリに存在するファイル名を、プリンタにリストします。

書　式

```
LFILES〔"デバイス名：〔パス名〕"〕
```

デバイス名：『ユーザーズマニュアル』の「ファイルについて」参照。
使用可能なデバイス名は次の通りです。
0：～3：、F0：～F3：、HD0：～HD3：、
MEM0：～MEM1：、EMM0：～EMM9：、
CAS：
パス名：『ユーザーズマニュアル』の「ファイルについて」参照

省略形

LF.

文　例

LFILES　"F1："
⇨8インチフロッピーディスク1のカレントディレクトリに存在するファイル名の一覧表をプリンタに打ち出します。

解　説

デバイス名で指定したデバイス内のカレントディレクトリに存在するファイル名をすべてプリンタにリストします。
使用方法は、FILESと同じです。

〈注　意〉

カセットに対してFILESコマンドを実行すると、デバイス名として"CAS0："が表示されますが、"CAS："と同様です。

参　照

2.1.18 FILES、『ユーザーズマニュアル』の「ファイルについて」

2.1.20 SAVE

機　能　プログラムをデバイスにセーブします。

書　式

```
SAVE 〔file〕〔,属性〕
```

file："デバイス名：〔パス名〕ファイル名"。
使用可能なデバイス名は次の通りです。
0：～3：、F0：～F3：、HD0：～HD3：、
MEM0：～MEM1：、EMM0：～EMM9：、
COM：、CAS：
『ユーザーズマニュアル』の「ファイルについて」参照。
属性：A（アスキー形式セーブの指定）またはP（プロテクトセーブの指定）

省 略 形　SA.

文　例　SAVE "CAS：TEST"
⇨メモリ内のプログラムをファイル名「TEST」としてカセットテープに中間コード形式で記録します。

解　説　SAVEを実行すると、メモリに記憶されているプログラムをデバイスに記録します。このことを「セーブする」といいます。

通常セーブは、中間コード形式[1]で行なわれますが、Aオプションをつけると、アスキー形式[2]で行なうことができます。ただし、デバイス名にCOM：を指定した場合は、Aオプションをつけなくても、アスキー形式のセーブが行なわれます。

1）中間コード形式……プログラムを、BASICの中間コードの形でデバイスへ記録する形式。内部表現は、機械語プログラムのバイナリー形式とほとんど同じです。

2）アスキー形式……プログラムを、1文字ずつキャラクタコードの形でデバイスへ記録する形式。キャラクタコードは、一般的にアスキーコードといいます。

アスキー形式のセーブは、デバイスにリストをとるようなもので、中間コード形式よりスペースを必要としますが、マージ（MERGE）をするためには、この形式でプログラムをセーブしておかなければなりません。

セーブするプログラムの秘密を守り、他人にLOADされないようにしたい場合は、ファイル名の後ろに「ファイル名；パスワード」の形式でパスワードをつけることができます。

パスワードはキャッシュカードにおける暗証番号のようなものです。

パスワードをつけてセーブしたファイルはパスワードを指定しなければ、ロードすることはできません。

"デバイス名：〔パス名〕ファイル名；パスワード"の"　"の中の指定は

31文字以内でなければなりません。

Ｐオプションを指定すると、同一ファイル名で他のファイルをセーブできません。一度、Ｐオプションで保護指定されたプログラムは、指定を解除できません。Ｐオプションをつけてセーブしたプログラムは、ＬＯＡＤ〔file〕すると、自動的に実行され、SHIFT ＋ BREAK キーで実行を停止できません。この場合、ＩＰＬリセットを押してください。ただし、プログラム中でＥＮＤ・ＳＴＯＰが実行されると終了・停止します。

参照

2.1.21 ＳＡＶＥＭ、2.1.26 ＭＥＲＧＥ、
『ユーザーズマニュアル』の「ファイルについて」

2.1.21 SAVEM

機能 **機械語プログラムをデバイスにセーブします。**

書式

```
SAVEM 〔file〕, a1, a2〔, a3〕
```

file："デバイス名：〔パス名〕ファイル名"。
使用可能なデバイス名は次の通りです。
0：～3：、F0：～F3：、HD0：～HD3：、
MEM0：～MEM1：、EMM0：～EMM9：、
COM：、CAS：
『ユーザーズマニュアル』の「ファイルの指定」参照。
a_1：セーブするプログラムの格納されている先頭アドレス。(スタートアドレス)
a_2：セーブするプログラムの格納されている最終アドレス。(エンドアドレス)
a_3：ロード後の実行開始アドレス。(エグゼックアドレス)

省略形 SA. M

文例 SAVEM "CAS：TEST", &H8000, &H81FF, &H8100

⇨メモリ内の機械語プログラムをアドレス&H8000から&H81FFまで、カセットテープにファイル名TESTとして記録し、アドレス&H8100からプログラムの実行を開始する指定をします。

解説 SAVEMコマンドを実行すると、メモリに記憶されている機械語プログラムをデバイスにセーブすることができます。

機械語プログラムをデバイスにセーブするときは、fileのあとにスタートアドレスa_1とエンドアドレスa_2を指定して、メモリのどこからどこまでをセーブするのか指定しなければなりません。このとき、その後ろに実行開始アドレスa_3を指定してセーブすると、あとで機械語プログラムをロードしたとき（LOADM実行時)、自動的にa_3から実行を始めることができます。a_3を指定しないでセーブすると、実行アドレスとしてスタートアドレスa_1が設定されます。

a_1、a_2、a_3のアドレスは、10進数、16進数、8進数、2進数のいずれを使って指定してもよいのですが、通常16進数を使います。

SEVEMはアドレスを指定することからもわかるように、単にメモリの内容をデバイスにバイナリー形式[1]でコピーするためのコマンドです。したがって、機械語プログラムばかりではなく、メモリのどの部分でもデータとしてセーブすることができます。

1）バイナリー形式………2進数のコードの形でデバイスへ記録する形式です。バイナリー（binary）とは、「2進、または2進法」という意味です。

参　照

2.1.20 SAVE、2.1.23 LOADM、
『ユーザーズマニュアル』の「ファイルについて」

2.1.22 LOAD

機能 指定したデバイスからプログラムを読み込みます。

書式

```
LOAD 〔file〕
```

file："デバイス名：〔パス名〕ファイル名"。
使用可能なデバイス名は次の通りです。
Ø：～3：、FØ：～F3：、HD0～HD3：、
MEMØ：～MEM1：、EMMØ：～EMM9：、
COM：、CAS：
『ユーザーズマニュアル』の「ファイルについて」参照。

省略形 LO.

文例 LOAD "CAS：TEST"
⇨ファイル名TESTのプログラムをカセットから読み込んでメモリに入れます。

解説 LOADコマンドを実行すると、デバイスにセーブされているプログラムをメモリに読み込みます。このことを「ロードする」といいます。

fileによってファイル名を指定すると、その名前のプログラムを探し出してロードします。ファイル名を省略すると、デバイスがカセットのときは、最初に現われたプログラムをロードし、他のデバイスのときは、「Bad file descripter」のエラーが出ます。

ファイルにパスワードがつけられているときは、ファイル名の後ろに"ファイル名；パスワード"の形式でパスワードを指定してロードするようにします。

このときパスワードなしでロードしようとすると「No pass word」のエラーが出ます。

カセットテープの場合、プログラムはなるべく1本に1つだけセーブするという習慣をつけておけば、プログラムをロードするとき、ファイル名を省略でき、時間もかかりません。

参照 2.1.23 LOADM、2.1.20 SAVE、
『ユーザーズマニュアル』の「ファイルについて」

2.1.23 ＬＯＡＤＭ

機能　デバイスから機械語プログラムを読み込みます。

書式

```
〔1〕LOADM 〔file〕〔, a〕
〔2〕LOADM 〔file〕〔, 〔a〕, R〕
```

file："デバイス名：〔パス名〕ファイル名"。
使用可能なデバイス名は次の通りです。
0：～3：、F0：～F3：、HD0：～HD3：、
MEM0：～MEM1：、EMM0：～EMM9：、
COM：、CAS：
『ユーザーズマニュアル』の「ファイルについて」参照。
a：メモリ内のロードスタートアドレス。
R：自動的に実行を開始する指定。

省略形　LO. M

文例　LOADM "CAS：TEST",, R
⇨ファイル名ＴＥＳＴの機械語プログラムをカセットから読み込んで、メモリに記憶後、ただちに実行します。

解説　デバイスにセーブされている機械語プログラムをメモリに読み込むには、ＬＯＡＤＭコマンドを使います。
fileによってファイル名を指定すると、その名のプログラムを探し出して読み込みます。ファイル名を省略すると、デバイスがカセット以外のときは、「Bad file descripter」のエラーが出ます。
ロードスタートアドレスaを指定すると、機械語プログラムが、メモリ内のそのアドレスからロードされます。aを指定しなければ、ＳＡＶＥＭで指定したスタートアドレスからロードされます。
Ｒオプションをつけると、機械語プログラムのロードが終わると自動的にＳＡＶＥＭのときに指定したスタートアドレスから実行を始めます。

参照　2.1.22 ＬＯＡＤ、2.1.21 ＳＡＶＥＭ、
『ユーザーズマニュアル』の「ファイルについて」

2.1.24 ＶＥＲＩＦＹ／ＬＯＡＤ？

機能

カセットテープにセーブしたプログラムが正しく記録されているかどうか調べます。

書式

```
〔1〕VERIFY file
〔2〕LOAD? file
```

file："ＣＡＳ：ファイル名"。
デバイス名は"ＣＡＳ："のみ使用可能。
『ユーザーズマニュアル』の「ファイルの指定」参照。

省略形

〔1〕ＶＥ.
〔2〕ありません。

文例

ＶＥＲＩＦＹ "ＣＡＳ：ＴＥＳＴ"
⇨カセットテープにセーブしたファイル名「ＴＥＳＴ」のプログラムが正しく記録されたかどうかチェックします。

解説

プログラムやデータをカセットにセーブするときは、他のデバイスよりやや信頼性の点で劣るので、あとで正しく記録されたかどうか確かめるためのデータ（チェックサム）がプログラムやデータといっしょにテープ上に書き込まれます。

セーブ実行後、ＶＥＲＩＦＹあるいはＬＯＡＤ？コマンドを実行すると、テープ上のプログラムファイルやデータファイルのチェックサムを新たに計算し、テープに記録されているチェックサムと照合を行なって、誤りなく記録されているかどうか調べます。

チェックサムが一致していれば正しく記録されていると判断し、「Ｏｋ」を表示してコマンド待ちの状態に戻ります。チェックサムが一致していなければ誤りがあると判断し、「Ｔａｐｅ ｒｅａｄ ｅｒｒｏｒ」のエラーを表示しますので、もう一度プログラムをセーブしなおすか、カセットテープを取り替えてください。

ｆｉｌｅ指定でファイル名を省略すると、最初に見つけたファイルの照合をします。

プログラムやデータのセーブ後はテープを巻き戻して、すぐにこのコマンドを実行する習慣をつけるとよいでしょう。

参照

2.1.20 ＳＡＶＥ、2.1.22 ＬＯＡＤ、
『ユーザーズマニュアル』の「ファイルについて」

サンプル
プログラム

```
10 ' VERIFY/LOAD? (例)
20 PRINT"VERIFY/LOAD? (例)"
30 END
SAVE"CAS:TEST"
Writing"TEST        .    "
Ok
REW
Ok
VERIFY"CAS:TEST"
Found  "TEST        .    "
Ok
REW
Ok
```

2.1

2.1.25 CHAIN

機能　デバイスからプログラムをロードして実行します。

書式

```
CHAIN 〔file〕
```

file：″デバイス名：〔パス名〕ファイル名″。
使用可能なデバイス名は次の通りです。
0：～3：、F0：～F3：、HD0～HD3：、
MEM0：～MEM1：、EMM0：～EMM9：、
COM：CAS：
『ユーザーズマニュアル』の「ファイルについて」参照。

省略形　CH.

文例　CHAIN ″1：TEST″
⇨メモリ中のプログラムの変数エリアの値を壊さないで、ファイル名「TEST」のプログラムを3または5インチフロッピーディスクのドライブ1からロードし実行します。

解説　メモリに存在するプログラムの変数の値を保存したまま、指定したデバイスからプログラムを読み込んで実行します。
RUNコマンドにファイルを指定して実行すると、変数の値をきれいにクリアしてから、新しいプログラムをロードして実行します。これに対してCHAINを実行すると、現在のプログラムの変数の値を保存した状態で次の新しいプログラムをロードして実行するので、保存された変数の値を新しいプログラム中で使用することができます。
メモリに納まり切らない大きなプログラムでも、いくつかのブロックに分割できれば、CHAINコマンドを使って、次々と実行することができます。

参照　2.1.3 RUN

サンプルプログラム

```
10 ' CHAINED PROGRAM (例1)
20 PRINT"CHAINED PROGRAM (例1)"
30 PRINT"SIN(X)=";SIN(X)
40 CHAIN"CAS:CHAIN2"
SAVE"CAS:CHAIN1"
Writing"CHAIN1        .    "
Ok
10 ' CHAINED PROGRAM (例2)
20 PRINT"CHAINED PROGRAM (例2)"
30 PRINT"COS(X)=";COS(X)
40 END
SAVE"CAS:CHAIN2"
Writing"CHAIN2        .    "
Ok
REW
Ok
10 ' MAIN PROGRAM (例)
20 INPUT"X(度)=";X
30 X=RAD(X)
40 CHAIN"CAS:CHAIN1"
RUN
X(度)=? 30
CHAINED PROGRAM (例1)
SIN(X)= .5
CHAINED PROGRAM (例2)
COS(X)= .8660254
Ok
```

2.1.26 MERGE

機　能　アスキー形式のプログラムを、現在メモリ内にあるプログラムに併合します。

書　式

```
MERGE 〔file〕
```

file："デバイス名：〔パス名〕ファイル名"。
使用可能なデバイス名は次の通りです。
0：～3：、F0：～F3：、HD0：～HD3：、
MEM0：～MEM1：、EMM0：～EMM9：、
COM：、CAS：
『ユーザーズマニュアル』の「ファイルについて」参照。

省略形　M.

文　例　MERGE "CAS：TEST"
⇨カセットテープからファイル名「TEST」のプログラムを読み込み、メモリ内のプログラムと組み合わせて1つにします。

解　説　MERGE（マージ）には、「混ぜ合わせる、合併する」などの意味があります。

このコマンドが実行されると、指定したデバイスに入っているプログラムが読み込まれ、現在メモリに存在するプログラムに組み合わされて1つのプログラムになります。このとき指定されたファイル名のデバイスに入っているプログラムは、アスキー形式でセーブされたものでなければなりません。アスキー形式でセーブされていないプログラムをマージ（MERGE）しようとすると「Bad file mode」のエラーが出ます。

アスキー形式でセーブされたプログラムを読み込むということは、入力デバイスがキーボードから現在プログラムの入っているデバイスにバトンタッチされたと考えることができます。したがって、マージを実行するとプログラムをキーボードから入力するのと同様の結果が得られることになります。たとえば、メモリ内のプログラムの行がファイル内の行と同じ番号のときは、メモリにあった行がファイル内の行に置き換えられてしまいます。

したがって、マージしたいプログラムをアスキー形式でセーブするときは、後でマージするときメモリ内のプログラムの行番号と重複する部分がないように、行番号をつけておく必要があります。

メモリ容量の制限から、MERGEするプログラムが大きすぎると正しくマージできないことがあります。

参　照　『ユーザーズマニュアル』の「ファイルについて」

サンプル
プログラム

```
10 ' MERGED PROGRAM (例)
20 INPUT"N=";N
90 END
SAVE"CAS:TEST",A
Writing"TEST     .   "
Ok
REW
Ok
NEW
Ok
LIST
Ok
30 S=SUM(N)
40 PRINT"SUM(N)=";S
LIST
30 S=SUM(N)
40 PRINT"SUM(N)=";S
Ok
MERGE"CAS:TEST"
Found  "TEST     .   "
Ok
LIST
10 ' MERGED PROGRAM (例)
20 INPUT"N=";N
30 S=SUM(N)
40 PRINT"SUM(N)=";S
90 END
Ok
```

2.1.27 MKDIR

機能

新しいディレクトリを作成します。

書式

```
MKDIR"デバイス名:〔パス名〕ディレクトリ名〔;パスワード〕"
```

デバイス名：使用可能なデバイス名は次の通りです。

0：～3：、F0：～F3：、HD0：～HD3：、
MEM0：～MEM1：、EMM0：～EMM9：

『ユーザーズマニュアル』の「ファイルについて」参照。

ディレクトリ名：13文字以内の文字列。『ユーザーズマニュアル』の「ファイルについて」参照。

パスワード：『ユーザーズマニュアル』の「ファイルについて」参照。

省略形

MK.

文例

MKDIR"0:営業;AYAKO"

⇨ディスクのドライブ0に"営業"というディレクトリを作成します。
そのとき、ディレクトリ参照のパスワードを"AYAKO"に設定します。

解説

MKDIRは、指定されるデバイスに、新しいディレクトリを作成するためのコマンドです。

パス名を省略すると、現在のカレントディレクトリの下に"営業"という名のディレクトリを作成します。

（例1） MKDIR"営業"
⇨現在のカレントディレクトリの下に"営業"という名のディレクトリを作成します。

（例2） MKDIR"/営業/鈴木"
⇨現在のカレントディレクトリがどこにあろうと、ルートディレクトリ下の"営業"ディレクトリのさらに下に"鈴木"という名のディレクトリを作成します。

（例3） MKDIR"/営業/佐藤;AYAKO"
⇨現在のカレントディレクトリがどこにあろうと、ルートディレクトリ下の"営業"ディレクトリのさらに下に"佐藤"という名のディレクトリを作成し、パスワードの"AYAKO"をつけます。

作成したディレクトリにファイルを作るときは、CHDIRでカレントディレクトリをそのディレクトリに移動してから行なってください。

参照

『ユーザーズマニュアル』の「ファイルについて」、2.1.28 CHDIR、2.1.29 RMDIR

2.1.28 CHDIR

機　能　カレントディレクトリを別のディレクトリに移動させます。

書　式

```
CHDIR"デバイス名：〔パス名〕ディレクトリ名〔;パスワード〕"
```

デバイス名：使用可能なデバイス名は次の通りです。

0：～3：、F0：～F3：、HD0：～HD3：、
MEM0：～MEM1：、EMM0：～EMM9：

パス名：『ユーザーズマニュアル』の「ファイルについて」参照。

ディレクトリ名：13文字以内の文字列。

パスワード：『ユーザーズマニュアル』の「ファイルについて」参照。

省略形　CHD.

文　例　CHDIR"0：/営業;AYAKO/在庫管理;MAKIKO/"

⇨ディスクのドライブ0のカレントディレクトリを"在庫管理"に移動します。

"営業"や"在庫管理"にパスワードがいるときは、それぞれにそのパスワードを指定する必要があります。

解　説

CHDIRは、指定されるデバイスのカレントディレクトリを、新しいディレクトリに移動するためのコマンドです。

パス名の頭のスラッシュ（/）を省略すると、現在のカレントディレクトリの下の新しいディレクトリに移動することができます。

移動後、作成したファイルはすべて、新しいディレクトリの中に作られます。

（例1）CHDIR"/営業"

⇨ルートディレクトリの下の"営業"という名のディレクトリに移動します。

（例2）CHDIR"鈴木;IQ200"

⇨現在のカレントディレクトリが"営業"とすると、その下の"鈴木"という名のディレクトリに移動します。パスワードがいるときは、そのパスワードを指定する必要があります。

（例3）CHDIR"/技術/田中;AYAKO"

⇨現在カレントディレクトリがどこに指定されていようと、ルートディレクトリ下の"技術"ディレクトリのさらに下の"田中"という名のディレクトリに移動します。

いったんカレントディレクトリが設定されると、再度、CHDIRを実行しない限り、そのカレントディレクトリは移動しません。したがって、ファイルへのアクセスをするときデバイス名を指定するだけで現在のカレントディレクトリの下のファイルに対してアクセスが行なわれます。

カレントディレクトリを移動する場合で、ルートディレクトリ側に戻りたい

場合は次のようにして移動します。

（例4） ＣＨＤＩＲ"／"

⇨ルートディレクトリに戻ります。

（例5） ＣＨＤＩＲ"．．"

⇨一つ上の階層にカレントディレクトリを移します。また"／"をつなげることでつなげた数だけ移ります。

（例6） ＣＨＤＩＲ"１："

⇨ドライブ１の現在のパス名を表示します。ディレクトリの移動はしません。

参　照　『ユーザーズマニュアル』の「ファイルについて」、2.1.27 ＭＫＤＩＲ、2.1.29 ＲＭＤＩＲ

2.1.29 RMDIR

機　　能　ディレクトリを消去します。

書　　式

```
RMDIR"デバイス名：〔パス名〕ディレクトリ名〔；パスワード〕"
```

デバイス名：使用可能なデバイス名は次の通りです。
　Ø：～3：、FØ：～3：、HDØ：～HD3：、
　MEMØ：～MEM1：、EMMØ：～EMM9：
パス名：『ユーザーズマニュアル』の「ファイルについて」参照。
ディレクトリ名：13文字以内の文字列。
パスワード：『ユーザーズマニュアル』の「ファイルについて」参照。

省 略 形　RM.

文　　例　RMDIR"Ø：／営業；AYAKO／在庫管理；MAKIKO"
⇨ディスクドライブØの"在庫管理"ディレクトリを消去します。"営業／"や"在庫管理／"にパスワードがついているときは、それぞれにそのパスワードを指定する必要があります。

解　　説　RMDIRは、指定されるデバイスの、いらなくなったディレクトリを消去するためのコマンドです。
パス名を省略すると、現在のカレントディレクトリの下の指定されたディレクトリを消去します。
指定したディレクトリにファイルが1つでも残っていると、「File already exists」のエラーが出て、ディレクトリの消去は行なわれません。ディレクトリを消去するときは、必ずその中にあるファイルのすべてをKILLで消去してからにしてください。
なお、現在のカレントディレクトリ自身を消去するには、パス名をルートディレクトリから指定する必要があります。

参　　照　2.1.27 MKDIR、2.1.28 CHDIR

2.1.30 NAME

機　能　ファイル名の更新をします。

書　式

```
NAME fileold AS filenew
```

fileold：旧ファイル。"デバイス名：〔パス名〕ファイル名"。
『ユーザーズマニュアル』の「ファイルについて」参照。
filenew：新ファイル。"ファイル名"。
『ユーザーズマニュアル』の「ファイルについて」参照。

省略形　NA.

文　例　NAME "1：TEST" AS "STAR"
⇨3または5インチフロッピーディスクのドライブ1にあるTESTという名のプログラムをSTARという名に変更します。

解　説　fileoldで指定したファイル名を、filenewのファイル名に変更します。
ここで使えるデバイス名は、次の通りです。
Ø：～3：……………………ディスク（3または5インチ版）
FØ：～F3：………………ディスク（8インチ版）
HDØ：～HD3：…………ハードディスク
MEMØ：～MEM1：……グラフィックメモリ
EMMØ：～EMM9：……外部メモリ

参　照　『ユーザーズマニュアル』の「ファイルについて」

2.1.31 COPY

機　能　ファイルをコピーします。

書　式

```
COPY fileold AS filenew
```

fileold：コピーするファイル。『ユーザーズマニュアル』の「ファイルについて」参照。

filenew：コピー後のファイル。『ユーザーズマニュアル』の「ファイルについて」参照。

文　例

ＣＯＰＹ"０：／営業／ＰＯ１"ＡＳ"１：／技術／田中／ＵＯ１"

⇨ディスクドライブ０番の"／営業／"ディレクトリの中の"ＰＯ１"という名のファイルを、ディスクドライブ１番の"／技術／田中／"ディレクトリの中に"ＵＯ１"という名でコピーします。

解　説

ＣＯＰＹは、各デバイス間でのファイルのコピーをするためのコマンドです。新しいファイル名を指定すると、違う名前のファイルとしてコピーすることができます。

コピーは同一のドライブ上でも行なうことができます。ただし、この場合は、異なるファイル名を指定しなければなりません。同じファイル名でコピーしようとすると、「Ｆｉｌｅ　ａｌｒｅａｄｙ　ｅｘｉｓｔｓ」のエラーが出ます。

異なるデバイスへコピーをするときには、同じファイル名を使うことができます。ただし、ランダムファイルがＯＰＥＮできるデバイスのみです。

なお、次のデバイスにコピーすることはできません。

カセットテープ（ＣＡＳ：）、画面（ＣＲＴ：、ＳＣＲ：）、キーボード（ＫＥＹ：）、プリンタ（ＬＰＴ：）、ＲＳ－２３２Ｃポート（ＣＯＭ：）

2.1.32 SET

機　能　ファイルに属性をつけます。

書　式

〔1〕SET　#n, 属性
〔2〕SET　file, 属性

n：ファイル番号。1～15の整数。MAXFILESで指定した番号まで。
file："デバイス名：〔パス名〕ファイル名"。『ユーザーズマニュアル』の「ファイルについて」参照。
属性："P"、"R"、"S"または""

文　例

SET　"0：TEST"，"P"
⇨3または5インチフロッピーディスクのドライブ0のTESTという名のファイルにライトプロテクトをかけます。
SET　"0：TEST"，""
⇨3または5インチフロッピーディスクのドライブ0のTESTという名のファイルの属性を取り除きます。

解　説

SETはファイルバッファまたはファイルに属性をつけるためのコマンドです。

属性を指定する文字には、"P"、"R"、"S"の3つがあります。このうち"P"はライトプロテクト（書き込み禁止）の属性をつけるための文字、"R"はリードアフタライト（書き込んだ後VERIFYする）の属性をつけるための文字、"S"はシークレット（FILESをとっても表示されない）の属性をつけるための文字です。

- "P"を設定するとファイルに対するPRINT#、WRITE#、PUTの書き込みが禁止され、KILL、SAVE、NAMEができなくなるので、ファイルを保護することができます。
- "R"を設定するとファイルに書き込みを行なった後すぐにVERIFYチェックをするため、正しく書き込みが行なわれていることを確認することができます。
- "S"を設定すると、FILESやLFILESを実行したとき、ファイル名のリスト中に表示されないため、ファイルのリストを小さく整理して表示したり、隠しファイルを作成することができます。

FILESでファイルのリストをとると、ライトプロテクトのついたファイルには*印が、リードアフタライトのついたファイルには#印がついて表示されます。シークレットのついたファイルは表示されませんのでこの属性をつけるときにはご注意ください。

属性を取り除くには、""（ヌルストリング）を指定します。

なお、カセットテープに対しては、使用できません。

参　照　2.1.18 FILES、2.1.19 LFILES

2.1.33 KILL

機　　能　デバイスに記録されたプログラムを消します。

書　　式

```
KILL file
```

file："デバイス名：〔パス名〕ファイル名"。
使用可能なデバイス名は次の通りです。
0：～3：、F0：～F3：、HD0：～HD3：、MEM0：～MEM1：、EMM0：～EMM9：
『ユーザーズマニュアル』の「ファイルについて」参照。

省 略 形　KI.

文　　例　KILL "1：TEST"
⇨3または5インチフロッピーディスクのドライブ1にあるTESTという名のプログラムを抹消します。

解　　説　KILLコマンドを実行すると、指定したファイル名のプログラムを消すことができます。
デバイス名に"CAS："を指定して、カセットテープに記録されたプログラムを消すことはできません。
※SETコマンドでPオプションをつけた場合はKILLできません。

参　　照　『ユーザーズマニュアル』の「ファイルについて」、2.1.32 SET

2.1.34　HDOFF

機　能

ハードディスクのヘッドを動作状態から解放し、待機状態にします。

書　式

```
HDOFF n
```

ｎ：ハードディスクのドライブ番号。０～３の整数。

文　例

ＨＤＯＦＦ　０

⇨ハードディスクの０番（"ＨＤ０："）のヘッドを待機状態にします。

解　説

ハードディスクのヘッドをデータのある所に置いたまま、ハードディスクの電源を切るとデータを壊してしまう恐れがあるので、このコマンドを実行してハードディスクのヘッドをデータのない所（レコード番号＆Ｈ９Ｅ４Ｂ）まで待機させてから電源を切るようにしてください。

ｎは、ハードディスクのドライブ番号で０～３が、"ＨＤ０："～"ＨＤ３："に対応します。

2.2 一般ステートメント

2.2.1 LET

機能　式の値を変数に代入します。

書式

〔LET〕 変数＝式

省略形　ＬＥＴは完全に省略することができます。

文例

ＬＥＴ　Ｘ＝１０
⇨数値変数Ｘに１０を代入します。
ＬＥＴ　Ｘ＄＝"パソコンテレビ"
⇨文字変数Ｘ＄に「パソコンテレビ」を代入します。

解説

ＬＥＴは、式の値を変数に代入するためのステートメントです。

代入の記号には等号（＝）を使い、等号の右側に式、左側に代入先の変数または配列が書かれます。ここでいう式には、単独の定数、変数、配列および関数も含みます。等号の左側が配列の場合も同様です。

右側の式の型は、左側の変数の型と一致していなければなりません。すなわち、式が数値型であれば変数は数値変数、式が文字型であれば変数は文字変数となります。

（正）　Ｘ＝Ａ　　Ｙ＝Ａ＊Ｘ＋１０　　　　Ｘ＄＝"＆Ｈ"＋Ａ＄
（誤）　Ｘ＄＝Ａ　Ｙ＄＝Ａ＊Ｘ＋"１０"　　Ｘ＝"＆Ｈ"＋Ａ＄

ＬＥＴは、初期のＢＡＳＩＣにおいて代入のステートメントに使われたもので、現在ではあってもなくてもよいステートメントとなっています。

サンプルプログラム

（例１）
```
10 ' LET <例1>
20 LET A%=3
30 LET A=3.14
40 LET A#=3.14159265359#
50 LET A$="HELLO"
60 LET B%=A%
70 LET B=A
80 LET B#=A#
90 LET B$=A$
100 PRINT B%,B,B#,B$
RUN
 3            3.14         3.14159265359        HELLO
Ok
```

２０～５０：変数に数値や文字列を代入しています。変数名の後ろの％、＃、＄はそれぞれ、変数が整数型、倍精度型、文字型であることを示しています。

６０～９０：変数の値を別の変数に代入しています。「数値精度の変換」を行わない限り、原則として両辺の変数の型は一致したものとなります。

(例２)

ＬＥＴは完全に省略することができるので、(例１)のプログラムは次の(例２)のように書くことができます。

```
10 ' LET (例2)
20 A%=3
30 A=3.14
40 A#=3.14159265359#
50 A$="HELLO"
60 B%=A%
70 B=A
80 B#=A#
90 B$=A$
100 PRINT B%,B,B#,B$
RUN
 3             3.14          3.14159265359          HELLO
Ok
```

2.2.2 REM

機　能　プログラムにコメントを入れます。

書　式

```
REM 〔コメント〕
```

コメント：注釈文。２５５文字までの文字列。

省略形　’（アポストロフィ）

文　例　１００ REM Sample Program 1985／09／14
⇨注釈文「Sample program 1985／09／14」を行番号１００としてプログラムに入れます。

解　説

ＲＥＭは、プログラムの中に注釈を書くためのステートメントで、実行にはかかわりありません。

ＲＥＭの右にマルチステートメントの形で他のステートメントを書くことはできません。注釈文の一部とみなされています。

（誤）１００ ＲＥＭ印刷：ＰＲＩＮＴ Ａ

（正）１００ ＰＲＩＮＴ Ａ：ＲＥＭ印刷

ＲＥＭは、アポストロフィ（’）で代用することができます。

（例）１００ ＰＲＩＮＴ Ａ：’印刷

「’」で始まるプログラム行はＬＩＳＴ＊やＬＬＩＳＴ＊によって、行番号と「’」を除いたコメントの部分だけ表示することができます。

ＬＩＳＴ＊ではプログラムのアポストロフィ（’）以後の部分のみを画面に表示できます。

また、ＬＬＩＳＴ＊では同様の部分をプリンタに出力できます。

サンプルプログラム

```
10 ' REM <例1>
20 REM*********************************
30 REM*                                *
40 REM* ピタゴラスの定理  1985/09/15   *
50 REM*                                *
60 REM*   A,B = 2辺の長さ              *
70 REM*   C   = 斜辺の長さ             *
80 REM*                                *
90 REM*********************************
100 REM データの入力
110 INPUT"A=";A
120 INPUT"B=";B
130 REM 計算
140 C=SQR(A*A+B*B)
150 REM 結果の表示
160 PRINT"C=";C
```

２０～９０：プログラムのタイトル文を書いています。タイトル文には、プログラムのタイトルのほか、プログラムの書かれた日付、プログラム中の変数の説明、プログラマの名前などを書きます。

１００,１３０,１５０：何をする部分なのか、簡単な注釈文を書いています。

REMは（例2）のように'(アポストロフィ）で代用することができます。

```
10 ' REM (例2)
20 '********************************
30 '*                              *
40 '* ピタゴラスの定理  1985/09/15  *
50 '*                              *
60 '*   A,B = 2辺の長さ            *
70 '*   C   = 斜辺の長さ           *
80 '*                              *
90 '********************************
100 ' データの入力
110 INPUT"A=";A
120 INPUT"B=";B
130 ' 計算
140 C=SQR(A*A+B*B)
150 ' 結果の表示
160 PRINT"C=";C
```

2.2.3 CLEAR／CLR

機　　能

すべての変数および配列をクリアします。

書　　式

〔1〕CLEAR
〔2〕CLR

省 略 形

〔1〕CLE.
〔2〕ありません。

文　　例

CLR
⇨変数と配列をすべてクリアします。

解　　説

プログラムの中のすべての変数および配列をクリアします。すなわち、変数の数値型のものは0に、文字型のものはヌルストリング（文字が何も入っていない状態）にし、配列はその定義を解除します。
CLEARは後ろにアドレスを指定すると、BASICの使用するメモリの最上位アドレスを設定するコマンドとして使用することができます。

参　　照

2.1.1 CLEAR／LIMIT

サンプルプログラム

```
10 ' CLEAR/CLR (例)
20 DIM E(20)
30 A%=12
40 B=1.23
50 C#=1.234567890123456#
60 D$="HELLO"
70 FOR I=0 TO 20
80 E(I)=I
90 NEXT
100 GOSUB 1000
110 CLEAR
120 PRINT:PRINT:PRINT"(CLEAR実行後)"
130 GOSUB 1000
140 END
150 '
1000 PRINT "A%=";A%
1010 PRINT "B =";B
1020 PRINT "C#=";C#
1030 PRINT "D$=";D$
1040 PRINT "E(I)=";
1050 FOR I=0 TO 20
1060 PRINT E(I);
1070 NEXT
1080 RETURN
```

30～90：変数と配列に値を代入しています。
100：サブルーチンを呼んで、変数と配列の値を表示します。
110：CLEAR文を実行して、変数と配列の値をクリアしています。
130：再びサブルーチンを呼んで、クリアした変数と配列の値を表示しています。

2.2

```
RUN
A%= 12
B = 1.23
C#= 1.234567890123456
D$=HELLO
E(I)= 0  1  2  3  4  5  6  7  8  9  10
11  12  13  14  15  16  17  18  19  20

(CLEAR実行後)
A%= 0
B = 0
C#= 0
D$=
E(I)= 0  0  0  0  0  0  0  0  0  0  0
Subscript out of range in 1060
Ok
```

2.2.4 DEFINT／DEFSNG／DEFDBL／DEFSTR

機能 変数の型を宣言します。

書式

```
〔1〕DEFINT 英字〔,英字,……〕
    DEFINT 英字ー英字
〔2〕DEFSNG 英字〔,英字,……〕
    DEFSNG 英字ー英字
〔3〕DEFDBL 英字〔,英字,……〕
    DEFDBL 英字ー英字
〔4〕DEFSTR 英字〔,英字,……〕
    DEFSTR 英字ー英字
```

英字：アルファベット1文字。A～Z、a～z。

省略形

〔1〕DEFINT……DEFI.
〔2〕DEFSNG……DEFS.
〔3〕DEFDBL……DEFD.
〔4〕DEFSTR……ありません。

文例

DEFINT　A－H
⇨AからHまでのアルファベットを整数型の変数とします。
DEFDBL　X，Y，Z
⇨アルファベットのX、Y、Zを倍精度型の変数とします。
DEFSTR　A－F，X
⇨アルファベットのAからFと、Xを文字型の変数とします。

解説

変数には、数値変数、文字変数、システム変数の3つがあり、数値変数は、さらに整数型、単精度型、倍精度型の3つのタイプに分けられます。このうち特殊なシステム変数を除いてそれぞれ変数名の後ろに属性文字（%、！、#、$）をつけて、変数の型を見分けます。

A%、SEISU%……整数型		数値変数
A！、SUU！……単精度型		
A#、BAI#……倍精度型		
A$、MOJI$……文字型		文字変数

ところで、プログラムによっては、ほとんどの変数を整数型として使いたい場合があります。そのとき、いちいち変数名の後ろに%をつけていたのでは面倒です。このことは、ほかの倍精度型、文字型の場合について同様にいえます。ただし、属性文字による指定がない場合は、変数は単精度型とみなされます。また［　ではじまる変数だけは例外となります。

このような場合に、DEFステートメントによって変数名の頭文字と型の対応を決定する宣言をして、属性文字を省略して使えるようにすることができま

す。

たとえば、Aはそのまま使うと単精度型の変数ですが、

DEFSTR A

と宣言すると、Aは文字型になり、A＄とわざわざ＄をつけなくても文字列を代入することができるようになります。また、ABC、AX1、A（1）など変数名の頭文字がAの変数、配列はすべて文字型とみなされます。

DEFINTは変数を整数型として、
DEFSNGは変数を単精度型として、
DEFDBLは変数を倍精度型として、
DEFSTRは変数を文字型として、

それぞれ宣言します。

宣言するときに、英字と英字の間をハイフン（－）でつなぐと、その範囲に含まれるアルファベットをすべて宣言したことになります。

型の宣言と属性文字による指定が異なる場合は、属性文字の方が優先されます。

参照　『ユーザーズマニュアル』の付録「数値精度の変換」

2.2.5　SWAP

機　能　**2つの変数の値を交換します。**

書　式

```
SWAP  x1, x2
```

x_1, x_2：値を交換する変数。数値変数、文字変数のいずれでもかまいません。
x_1とx_2の変数の型は一致していなければなりません。

省略形　SW.

文　例　SWAP　A, B
⇨数値変数Aの値を数値変数Bに、数値変数Bの値を数値変数Aに入れて、値を交換します。

解　説　SWAPステートメントは、2つの変数の値を交換します。たとえば、変数Aの値が10、Bの値が20のとき、

SWAP　A, B

を実行すると、Aが20、Bが10となります。

これを従来の代入だけで行なおうとすると、次の3つのステートメントが必要となります。

X＝A
A＝B
B＝X

変数は、数値変数、文字変数どちらでもかまいませんが、2つの変数の型は一致していなければなりません。

なお、VDIM変数の値と普通の変数の値との交換はできません。

(例)
```
10  WIDTH 40, 25, 0
20  OPTION SCREEN 1
30  VDIM CLEAR
40  VDIM A
50  A=2
60  B=3
70  SWAP A, B
80  PRINT A, B
90  END
```
これをRUNすると、70行で「Type mismatch」のエラーがでます。

サンプルプログラム


```
10 ' SWAP (例)
20 KMODE 1
30 OPTION BASE 1
40 DIM A(20)
50 READ N
60 PRINT"サンプル数=";N
70 PRINT"入力データ"
80 FOR I=1 TO N
90 READ A(I)
100 PRINT A(I);
110 NEXT
120 FOR I=1 TO N
130 FOR J=I TO N
140 IF A(I)<A(J) THEN SWAP A(I),A(J)
150 NEXT
160 NEXT
170 PRINT
180 PRINT"出力データ"
190 FOR I=1 TO N
200 PRINT A(I);
210 NEXT
220 DATA 10
230 DATA 21,35,54,16,77,43,95,88,61,0
```

５０：２２０のＤＡＴＡ文からサンプル数を読み込んでいます。

８０～１１０：数値データを配列へ読み込み、表示しています。

１２０～１６０：読み込んだ数値データを大きい順に並び換えています。

１９０～２１０：並び換えた結果を表示しています。

２２０～２３０：数値データの並びです。

```
RUN
サンプル数= 10
入力データ
 21  35  54  16  77  43  95  88  61  0
出力データ
 95  88  77  61  54  43  35  21  16  0
Ok
```

2.2.6 STOP

機　能　プログラムの実行を止めます。(⇔CONT)

書　式

```
STOP
```

省略形　S.

文　例　STOP

⇨プログラムの実行を停止します。

解　説

プログラムの実行を途中で止めるには、止めたい箇所にSTOPステートメントを入れます。実行がSTOPのところに来ると、ビープ音を鳴らし画面に次のメッセージを表示してコマンドの入力待ちの状態になります。

Break in 行番号

ここで、行番号はSTOPの入っているプログラムを示しています。

STOPで止めたプログラムは、CONT⏎と入力することによって、次のステートメントから実行を再開することができます。

このコマンドは、プログラムを一時停止して、変数の値をチェックしたいときなどに使うと便利です。

参　照　2.1.4 CONT、2.2.7 END

サンプルプログラム

```
10 ' STOP <例>
20 INPUT X
30 Y=X*RND
40 STOP
50 Z=Y*100
60 PRINT Z
```

このプログラムを実行すると、40で実行が停止しますが、CONT⏎と入力すれば50から実行を再開することができます。

```
RUN
? 10                 → 適当な値を入力する。
Break in 40          → STOP文でプログラムの実行が止まる。
Ok.
? Y                  → 停止した時点でのYの値を確認する。
 7.463764
Ok.
CONT                 → CONT文を入力して実行を再開する。
 746.3764
Ok
```

2.2.7 END

2.2

機　能　　**プログラムの実行を終了します。**

書　式

```
END
```

文　例　　ＥＮＤ

⇨プログラムを終了し、全てのファイルをクローズします。

解　説

　ＥＮＤステートメントを実行すると、プログラムは終了し、すべてのファイルをクローズして、コマンドの入力待ちの状態に戻ります。

　ＥＮＤは、プログラム中のどこにでも入れることができます。しかし、まったく書かなくてもプログラムの最終行にくれば自然に実行を終了します。したがって、プログラムの最終行のＥＮＤは省略してもかまいません。

参　照　　2.2.6 ＳＴＯＰ、『ユーザーズマニュアル』の「ファイルについて」

サンプルプログラム

（例１）

```
10 ' END (例1)
20 CLS
30 KMODE 1
40 A$=INKEY$
50 IF A$=" " THEN 80
60 GOSUB 100
70 GOTO 40
80 PRINT"終了"
90 END
100 LOCATE 0,0:PRINT"時刻="；TIME$
110 RETURN
```

４０～７０：スペースキー・キーの入力を待ち、入力があれば８０へジャンプ、なければサブルーチンを呼んで現在時刻を表示します。

９０：プログラムの終わりを示すＥＮＤです。

１００～１１０：画面の左上隅に現在時刻を表示するサブルーチンです。

（例２）

```
10 ' END (例2)
20 CLS
30 KMODE 1
40 A$=INKEY$
50 IF A$=" " THEN 80
60 GOSUB 100
70 GOTO 40
80 PRINT "終了"
90 '
100 LOCATE 0,0:PRINT "時刻="；TIME$
110 RETURN
```

（例２）のように９０のＥＮＤを取って実行すると、スペース・キーを押した時点で、１００以降のサブルーチンに抜けてしまい、「ＲＥＴＵＲＮ　without　ＧＯＳＵＢ　in　１１０」のエラーが出ます。

2.2.8 OPTION BASE

機　能　**配列の添字の最小値を宣言します。**

書　式

```
OPTION BASE n
```

n：0または1

省略形　OP. B.

文　例　OPTION BATE 1

⇨配列の添字の最小値が1となります。

解　説　BASIC起動時、配列の添字は0から使えるように設定されています。

たとえば、

```
100 DIM A(100)
```

と配列宣言すれば、

A(0)、A(1)、A(2)、……、A(100)

の配列を使うことができます。

しかし、

```
100 OPTION BASE 1
110 DIM A(100)
```

というように、配列宣言の前にOPTION BASEステートメントを宣言すれば、添字の最小値が1となって、

A(1)、A(2)、……、A(100)

の配列が使えるようになります。

OPTION BASEステートメントは、必ずDIMステートメントで配列宣言する前に書かなければなりません。

ただし、最小値0から使用する場合、OPTION BASE 0は宣言しなくてもかまいません。また、一度宣言されると再宣言できません。CLEAR、CLR、RUN、NEW、NEWONによって、この宣言は解除されます。

CLEAR、CLR、RUN、NEW、NEWONを実行すると、BASIC起動時の状態に戻り、配列の添字の最小値は0となります。

参　照　2.2.9 DIM、2.2.10 VDIM、『ユーザーズマニュアル』の「配列」

サンプルプログラム

```
10 ' OPTION BASE (例1)
20 OPTION BASE 0
30 DIM A(10)
40 FOR I=0 TO 10
50 A(I)=I
60 PRINT A(I);
70 NEXT
80 END
RUN
 0  1  2  3  4  5  6  7  8  9  10
Ok
```

２０，３０：配列添字の最小値を０に設定して、配列宣言します。

このようにして宣言した配列は、

A(０)，A(１)，……，A(１０)

を使用することができます。

```
10 ' OPTION BASE (例2)
20 OPTION BASE 1
30 DIM A(10)
40 FOR I=1 TO 10
50 A(I)=I
60 PRINT A(I);
70 NEXT
80 END
RUN
 1  2  3  4  5  6  7  8  9  10
Ok
```

２０，３０：配列添字の最小値を１に設定して、配列宣言します。

こうして宣言した配列は、

A(１)，A(２)，……，A(１０)

を使用することができます。

これは、配列のデータの順序と添字の値が一致するという利点をもっています。

2.2.9 DIM

機能　配列宣言をメインメモリ上に行ないます。

書式

```
DIM 配列名 (m1 〔, m2, ……〕)
        〔, 配列名 (n1 〔, n2, ……〕), ……〕
```

配列名：配列の名前。変数名のつけ方と同じです。
m_1、m_2、n_1、n_2…：配列のサイズ。ただし1～255の整数。

文例　DIM A (2, 3)
⇨Aという配列名の2次元配列を宣言します。

解説

配列の名前と次元数、および添字のサイズを設定します。

1つのDIM文で複数個の配列を同時に宣言できます。メモリの範囲内で255次元配列までの設定が可能です。(1行の入力文字数の制限から実際には255次元まで指定できません)

(例1) 1 0 0　DIM A (1 0 0)
　　A (0)、A (1)、A (2)、……、A (1 0 0)
　　の配列を宣言しています。

(例2) 1 0 0　DIM A (1 0, 1 0)
　　A (0, 0)、A (0, 1)、……、A (0, 1 0)
　　A (1, 0)、A (1, 1)、……、A (1, 1 0)
　　……………………………………………………
　　A (1 0, 0)、A (1 0, 1)、……、A (1 0, 1 0)
　　の配列を宣言しています。

(例3) 1 0 0　MAX＝1 0 0
　　1 1 0　DIM A (MAX)
　　A (0)、A (1)、A (2)、……、A (1 0 0)
　　の配列を宣言しています。

配列宣言をすると、数値型配列には0、文字型配列にはヌルストリングが入り、すべての配列はクリアされた状態となります。

参照　『ユーザーズマニュアル』の「配列」、2.2.8 OPTION BASE、2.2.10 VDIM

サンプル
プログラム

2.2

```
10 ' DIM (例)
20 DIM A(2),B(2,2),C(2,2,2)
30 FOR I=0 TO 2
40 A(I)=I
50 PRINTUSING"####";A(I);
60 NEXT
70 PRINT:PRINT
80 FOR I=0 TO 2
90 FOR J=0 TO 2
100 B(I,J)=I*10+J
110 PRINTUSING"####";B(I,J);
120 NEXT
130 PRINT
140 NEXT
150 PRINT
160 FOR I=0 TO 2
170 FOR J=0 TO 2
180 FOR K=0 TO 2
190 C(I,J,K)=I*100+J*10+K
200 PRINTUSING"####";C(I,J,K);
210 NEXT
220 PRINT
230 NEXT
240 NEXT
RUN
   0   1   2

   0   1   2
  10  11  12
  20  21  22

   0   1   2
  10  11  12
  20  21  22
 100 101 102
 110 111 112
 120 121 122
 200 201 202
 210 211 212
 220 221 222
Ok
```

２０：数値型配列を宣言しています。

３０～６０：１次元配列への数値の代入と表示をします。

８０～１４０：２次元配列への数値の代入と表示をします。

１６０～２４０：３次元配列への数値の代入と表示をします。

2.2.10 VDIM

機　　能　変数エリアの確保および配列宣言をグラフィックメモリに行ないます。

書　　式

```
VDIM 変数名〔(m1〔, m2, ……〕)〕〔, 変数名〔(n1〔, n2, ……〕)〕,
      ……〕
```

変数名：配列名。変数名のつけ方と同じです。
m_1, m_2, n_1, n_2, ……：配列のサイズ。１～２５５の整数。

省 略 形　V.

文　　例　VDIM A (1 0 0)
⇨配列Ａ（0）～Ａ（1 0 0）がグラフィックメモリ１に割り当てられます。

解　　説

配列と次元数とサイズを設定し、同時にグラフィックメモリ上にその配列領域を割り当てます。また、サイズの指定を省略した場合は、配列変数ではなく通常の変数がグラフィックメモリ上に割り当てられます。

配列領域や変数領域は、グラフィックメモリ１に割り当てられます。

ＶＤＩＭを使用できるスクリーンモードはＯＰＴＩＯＮ　ＳＣＲＥＥＮステートメントによって、

　　ＯＰＴＩＯＮ　ＳＣＲＥＥＮ　１
　　ＯＰＴＩＯＮ　ＳＣＲＥＥＮ　２

のいずれかが実行されている場合です。

これ以外のスクリーンモードでＶＤＩＭを実行すると、「Ｂａｄ　ｓｃｒｅｅｎ　ｍｏｄｅ」のエラーとなります。ＶＤＩＭにより確保された変数領域をクリアする場合には、ＶＤＩＭ　ＣＬＥＡＲを用います。ＣＬＥＡＲ、ＣＬＳなどでは消去されません。

ＶＤＩＭの添字はＯＰＴＩＯＮ　ＢＡＳＥ命令により、影響されます。

また、ＶＤＩＭで割り当てた変数や配列をプログラム中のＦＯＲ－ＮＥＸＴループのループ変数、ＧＥＴ＠、ＰＵＴ＠、ＬＳＥＴ、ＲＳＥＴ、ＭＩＤ＄による代入、およびＦＩＥＬＤステートメント中で使うことはできません。

参　　照　2.2.11 ＶＤＩＭ　ＣＬＥＡＲ、2.2.9 ＤＩＭ、
2.8.1 ＯＰＴＩＯＮ　ＳＣＲＥＥＮ、2.2.12 ＥＲＡＳＥ

サンプル
プログラム

2.2

```
10 ' VDIM (例)
15 WIDTH ,,0
20 OPTION SCREEN 1:VDIM CLEAR
30 VDIM A(2),B(2,2),C(2,2,2)
40 FOR I=0 TO 2
50 A(I)=I
60 PRINT USING"####";A(I);
70 NEXT
80 PRINT:PRINT
90 FOR I=0 TO 2
100 FOR J=0 TO 2
110 B(I,J)=I*10+J
120 PRINT USING"####";B(I,J);
130 NEXT
140 PRINT
150 NEXT
160 PRINT
170 FOR I=0 TO 2
180 FOR J=0 TO 2
190 FOR K=0 TO 2
200 C(I,J,K)=I*100+J*10+K
210 PRINT USING"####";C(I,J,K);
220 NEXT
230 PRINT
240 NEXT
250 NEXT
260 VDIM CLEAR
```

２０：グラフィックメモリ１を変数エリアとして使用する宣言をします。

３０：グラフィックメモリ１に配列を割り当てます。

４０～７０：１次元配列への数値の代入と表示を行ないます。

９０～１５０：２次元配列への数値の代入と表示を行ないます。

１７０～２５０：３次元配列への数値の代入と表示を行ないます。

2.2.11 VDIM CLEAR

2.2

働き　VDIMで定義したすべての変数および配列をクリアします。

書式

```
〔1〕VDIM CLEAR
〔2〕VDIM CLEAR a1, a2
```

a_1：VDIMの先頭アドレス。&H4000～&HFFFF。
a_2：VDIMのエンドアドレス。&H4000～&HFFFF。

省略形　V. CLE.

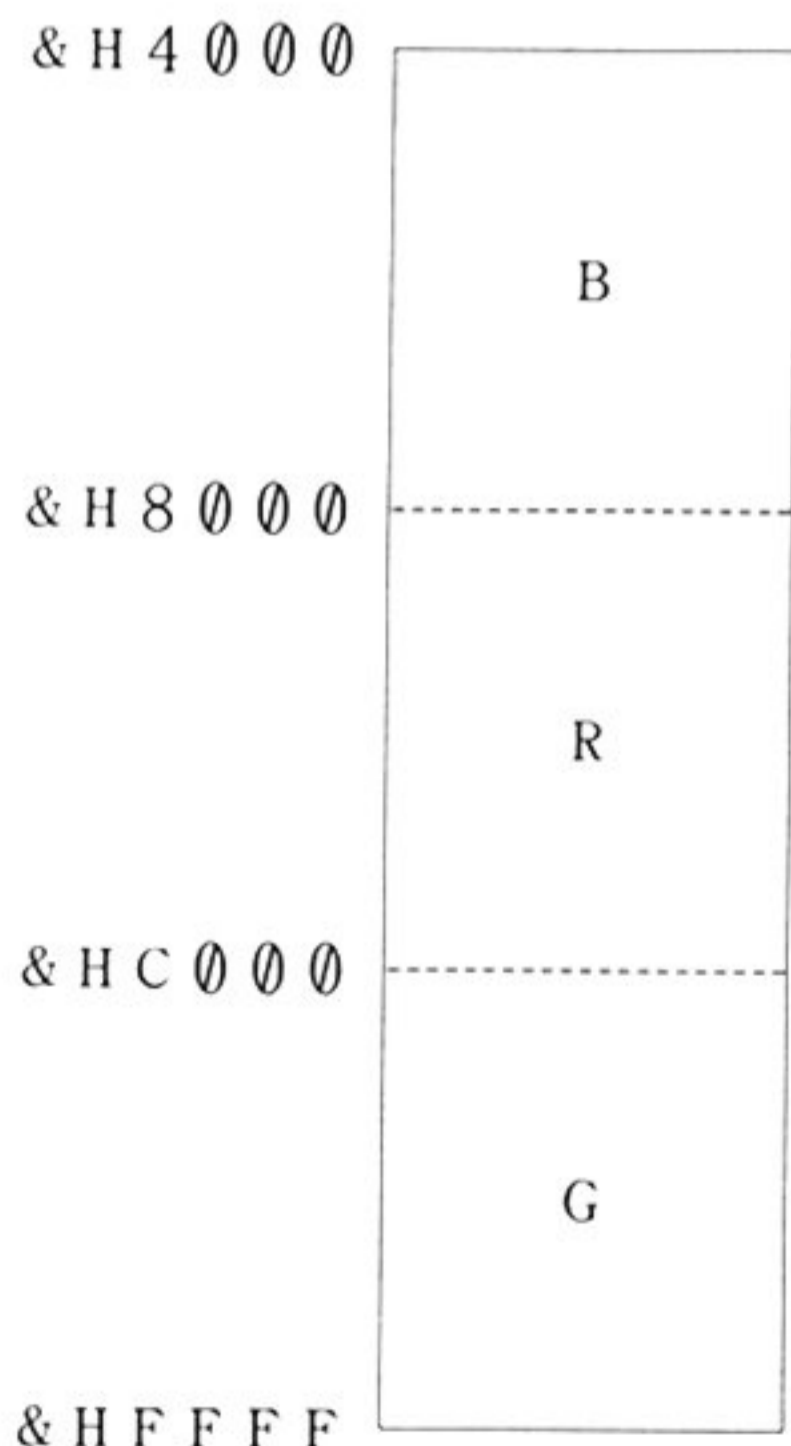
文例　VDIM CLEAR &H4000, &H7FFF
⇨グラフィックメモリ上の&H4000から&H7FFFをVDIMの変数定義エリアとして確保します。

解説

VDIM CLEARを実行すると、グラフィックメモリ上のすべての変数および配列をクリアします。

後ろにアドレスで範囲を指定すると、その領域がVDIMで使用する領域として確保されます。

これによって、1つのグラフィックメモリを同時にグラフィック表示用と変数エリア用として使用することができます。

OPTION SCREEN1またはOPTION SCREEN2を指定すると、グラフィックメモリ1が変数エリアとして使えます。下図は、グラフィックメモリ1のメモリマップです。

たとえばＶＤＩＭ　ＣＬＥＡＲ　＆Ｈ４０００，＆Ｈ７ＦＦＦと指定した場合、Ｂの領域がＶＤＩＭで使用するエリアとして確保されます。残りのＲ、Ｇのエリアは、表示用として使用できます。したがってＲ（赤），Ｇ（緑）は、４００のライン表示ができます。

（例）１０　ＷＩＤＴＨ,，０
　　　２０　ＯＰＴＩＯＮ　ＳＣＲＥＥＮ　１
　　　３０　ＶＤＩＭ　ＣＬＥＡＲ　＆Ｈ４０００，＆Ｈ４ＦＦＦ
　　　４０　ＷＩＤＴＨ,，１

ＶＤＩＭ　ＣＬＥＡＲのa_1，a_2を指定した場合（グラフィックメモリ１全体またはa_2＝＆ＨＦＦＦＦを指定した時を除く）では、（例）のように実行しても、「Ｂａｄ　ｓｃｒｅｅｎ　ｍｏｄｅ」のエラーは出ません。（（例）では、＆Ｈ５０００～＆ＨＦＦＦＦを表示用として使えると判断します。）

したがって、表示用として使用する場合、画面モードによって、ＶＤＩＭエリアを壊す場合がありますので注意して使用してください。

参　照

2.2.10 ＶＤＩＭ，『ユーザーズマニュアル』の「ディスプレイモード」と「メモリマップ」と「フリーエリアについて」

サンプルプログラム

```
10 ' VDIM CLEAR (例)
20 WIDTH80,25,1,2:CLS0
30 OPTIONSCREEN2
40 VDIMCLEAR &H4000,&HBFFF
50 SCREEN0,0,3:'CANVAS0,0,7:LAYER1,3,4,2

60 VDIM A(1000),B(1000)
70 LINE(0,0)-(639,399),PSET,1,B:CIRCLE(3
20,200),90
80 FOR I=0 TO 1000
90 A(I)=I:PRINT A(I);
100 NEXT
110 END
```

2.2.12 ERASE

機　能　配列を消去します。

書　式

```
ERASE A1 〔, A2, ……〕
```

Ａ１，Ａ２，……：配列名。

省略形　ＥＲ.

文　例　ＥＲＡＳＥ　Ｘ，Ｙ

⇨配列ＸとＹを消去します。

解　説　ＥＲＡＳＥは、指定した配列を消去するためのステートメントです。消去した配列はＤＩＭやＶＤＩＭで再び宣言することができます。

ＥＲＡＳＥで消去せずに同じ配列を宣言しようとすると、「Ｄｕｐｌｉｃａｔｅ Ｄｅｆｉｎｉｔｉｏｎ」のエラーが出ます。

参　照　2.2.9 ＤＩＭ、2.2.10 ＶＤＩＭ

サンプルプログラム

```
10 DIM A(100),B(100),C(100)
20 ERASE A,B
30 DIM A(150)
40 DIM B(200)
50 DIM C(250)
RUN
Duplicate definition in 50
Ok
```

２０行によって配列Ａ、Ｂが消されます。
したがって３０行、４０行が実行されていますが、５０行は、１０行で宣言されているのでエラーとなります。

2.2.13 LABEL

機　　能　プログラム行にラベルをつけます。

書　　式

```
LABEL L$
```

Ｌ＄：ラベル名。"文字列"。２５５文字までの文字列。

省 略 形　ＬＡ．

文　　例　ＬＡＢＥＬ"印刷"
⇨プログラム中にラベル名"印刷"をつけます。

解　　説　このＢＡＳＩＣでは、ＧＯＴＯ、ＧＯＳＵＢステートメントなどのジャンプ先として行番号の代りに、ラベルを指定することができます。ラベル名は、２５５文字以内であればどんな行番号でもかまいません。
プログラム中に同じラベルがある場合は最初のラベルが優先されます。
ラベル名を使えるステートメントは、以下に示す通りです。
ＧＯＴＯ、ＧＯＳＵＢ、ＩＦ－ＴＨＥＮ－ＥＬＳＥ、ＯＮ　～　の形式の各ステートメント（ただし、ＯＮ　ＫＥＹ　ＧＯＳＵＢには使えません）
ＲＥＳＵＭＥ、ＲＥＴＵＲＮ、ＲＥＳＴＯＲＥ

サンプル プログラム　このプログラムは時刻を表示させておき、スペースキーが押された時"終了"を表示して終るものです。

```
10 ' LABEL (例)
20 CLS
30 LABEL"キー入力"
40 A$=INKEY$
50 IF A$=" " GOTO"プログラム終了"
60 GOSUB"時刻表示"
70 GOTO"キー入力"
100 LABEL"プログラム終了"
110 PRINT"終了"
120 END
1000 LABEL"時刻表示"
1010 LOCATE 0,0:PRINT "時刻=";TIME$
1020 RETURN
```

３０：ラベル名"キー入力"が設定されています。このラベル名は、７０のＧＯＴＯ文のジャンプ先として参照されています。
１００：ラベル名"プログラム終了"が設定されています。これは、５０のＩＦ文の分岐先として参照されています。
１０００：ラベル名"時刻表示"が設定されています。これは、サブルーチンの入口となっていて、６０のＧＯＳＵＢ文から呼び出されます。
１０００～１０２０：現在の時刻を画面左上隅に表示するサブルーチンです。

2.2.14 GOTO

2.2

機　　能　**指定した行番号、ラベルにジャンプします。**

書　　式

```
〔1〕GOTO n
〔2〕GOTO L$
```

n：行番号。１～６５５３４の整数。
L＄：ラベル名。"文字列"。ただし２５５文字までの文字列。
　　LABELステートメントで定義しているもの。

省 略 形　G.

文　　例　GOTO　１５０
⇨無条件に行番号１５０へジャンプします。
GOTO"**印刷**"
⇨無条件にLABEL"**印刷**"のある行へジャンプします。

解　　説　GOTOは指定した行番号かラベルにジャンプするためのステートメントです。ジャンプ先は前後いずれでもかまいません。
エラーなどでプログラムが止まったとき、ダイレクトモードでGOTOステートメントを入力して、変数の状態を保持したまま、途中の行から実行を再開することができます。

サンプルプログラム

```
10 ' GOTO (例1)
20 CLS
30 A$=INKEY$
40 IF A$=" " THEN PRINT"終了":END
50 LOCATE 0,0:PRINT "時刻=";TIME$
60 GOTO 30
```

（例１）
３０，４０：キー入力がスペースのときはプログラムの実行を終了しますが、
　　　　　それ以外は５０を実行します。
６０：GOTO３０で３０番に戻ります。

```
10 ' GOTO (例2)
20 CLS
30 LABEL"キー入力"
40 A$=INKEY$
50 IF A$=" " THEN PRINT"終了":END
60 LOCATE 0,0:PRINT "時刻=";TIME$
70 GOTO"キー入力"
```

（例２）
４０，５０：キー入力がスペースのときはプログラムの実行を終了しますが、
　　　　　それ以外は６０を実行します。
７０：GOTO"キー入力"で３０のラベルのところに戻ります。

2.2.15 GOSUB

機　能　　サブルーチンを呼び出します。

書　式

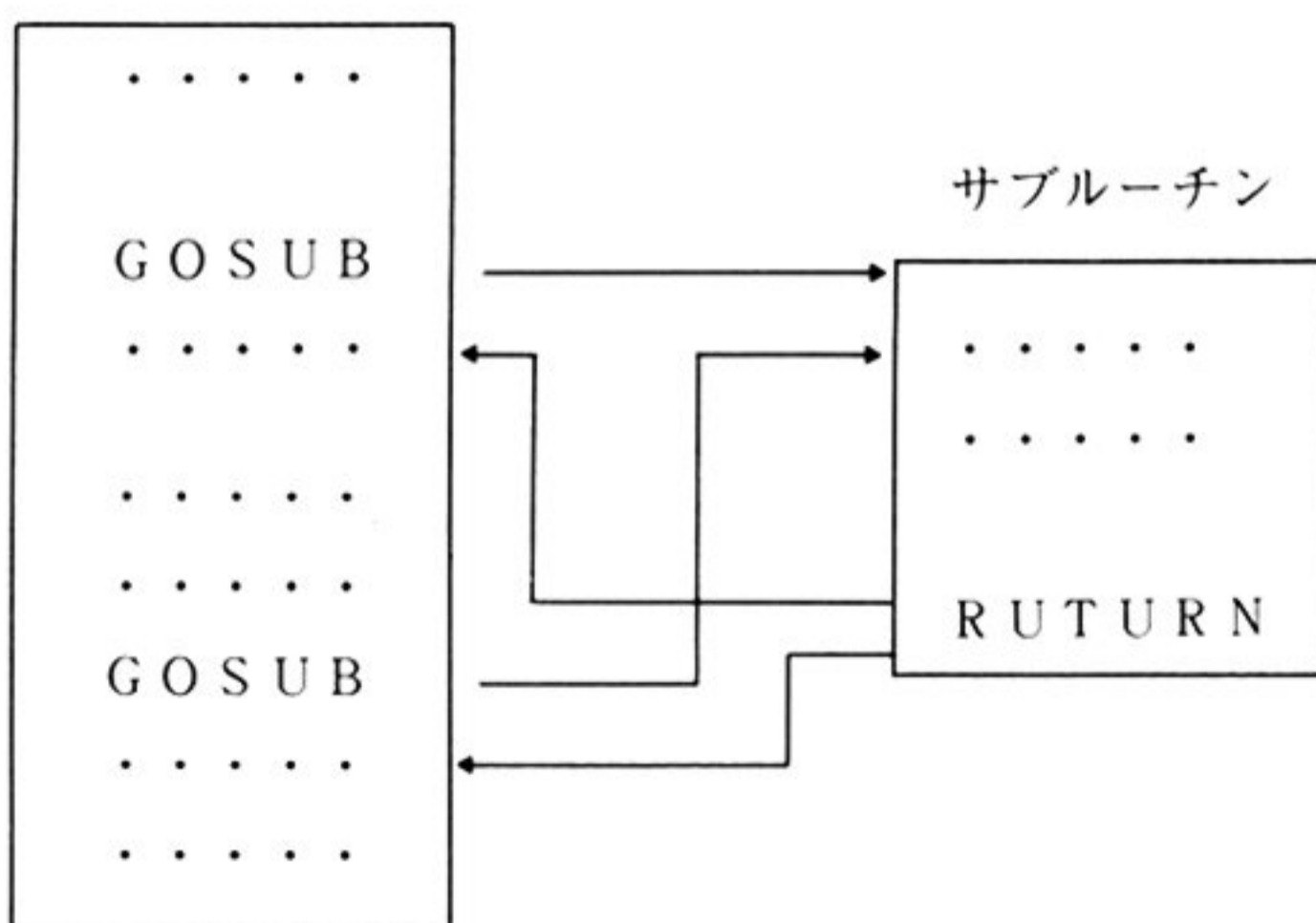
〔1〕GOSUB n
〔2〕GOSUB L$

n：サブルーチンの先頭行番号。
L＄：ラベル名。"文字列"。２５５文字までの文字列。
ＬＡＢＥＬステートメントで定義しているもの。

省略形　　ＧＯＳ．
(「ＧＯＳＵＢ　Ｌ＄」と書く代わりに「ＧＯＳ．Ｌ＄」と書くこともできます。)

文　例　　ＧＯＳＵＢ　１０００
⇨行番号１０００から始まるＢＡＳＩＣプログラムのサブルーチンを呼び出し、ＲＥＴＵＲＮまで実行します。

解　説　　プログラム中、何箇所かで同じ処理を行なう必要がある場合、その処理を１つのサブプログラムとして作っておき、プログラム（メインルーチン）のしかるべき箇所から呼び出すようにすることができます。メインルーチンから呼び出すことのできるサブプログラムのことを「サブルーチン」といいます。

サブルーチンは、ＧＯＳＵＢステートメントを使って呼び出すことができます。

ＧＯＳＵＢの後ろの行番号やラベル名は、サブルーチンの開始行を表わしています。サブルーチンの最後には必ずＲＥＴＵＲＮをつけます。

サブルーチンの中にＧＯＳＵＢを書いて、さらに別のサブルーチンを呼び出すこともできます。

参照　2.2.16 RETURN、1.11「関数とサブルーチン」

サンプルプログラム

```
10 ' GOSUB (例)
20 ' メインルーチン
30 FOR I=1 TO 10
40 GOSUB 1000
50 NEXT
60 FOR I=10 TO 1 STEP -1
70 GOSUB 1000
80 NEXT
90 PRINT
100 END
1000 ' サブルーチン
1010 BEEP
1020 FOR J=1 TO I
1030 PRINT CHR$(&H41);
1040 NEXT
1050 PRINT
1060 RETURN

RUN
A
AA
AAA
AAAA
AAAAA
AAAAAA
AAAAAAA
AAAAAAAA
AAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAA
AAAAAAAA
AAAAAAA
AAAAAA
AAAAA
AAAA
AAA
AA
A

Ok
```

２０～１００：メインルーチン

３０～５０：サブルーチンを１０回呼んで実行します。

６０～８０：ＦＯＲ－ＮＥＸＴループは異なりますが、３０～５０と同様、サブルーチンを１０回呼んで実行します。

１００：メインルーチンの終わりを示しています。

１０００～１０６０：サブルーチン

１０１０：ビープ音を鳴らします。したがって、サブルーチンが呼ばれるたびにビープ音が鳴ることになります。

１０２０～１０４０：キャラクタの表示をします。

１０６０：サブルーチンの出口です。

2.2.16 RETURN

2.2

機　　能　サブルーチンからの復帰をします。

書　　式

〔1〕RETURN　〔n〕
〔2〕RETURN　〔L$〕

n：行番号。復帰先の行番号。1～65534の整数。
L$：ラベル名。"文字列"。復帰先のラベル。
LABELステートメントで定義したもの。

省 略 形　RE.

文　　例　RETURN　2００
⇨サブルーチンの終わりを示し、メインルーチンの行番号2００へ戻り実行します。

解　　説　RETURNは、GOSUBで呼ばれたサブルーチンの最後につけて、もとの所に戻るときに使います。
プログラムの流れがGOSUBの所に来ると、サブルーチンにジャンプして、そこのプログラムを実行し、RETURNに達すると、GOSUBの次のステートメントに戻ります。
RETURNの後ろの部分に行番号かラベルを指定すると、その行に戻ります。

参　　照　2.2.15 GOSUB

2.2.17 IF－THEN－ELSE

機能　**条件式を判断して、分岐を行ないます。**

書式

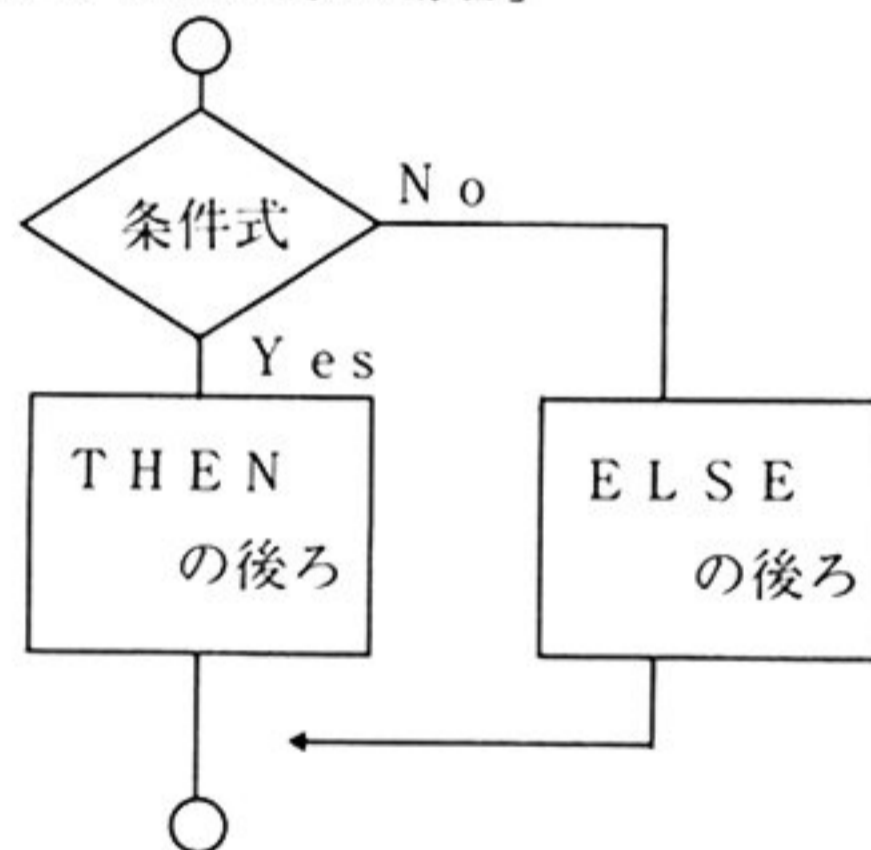
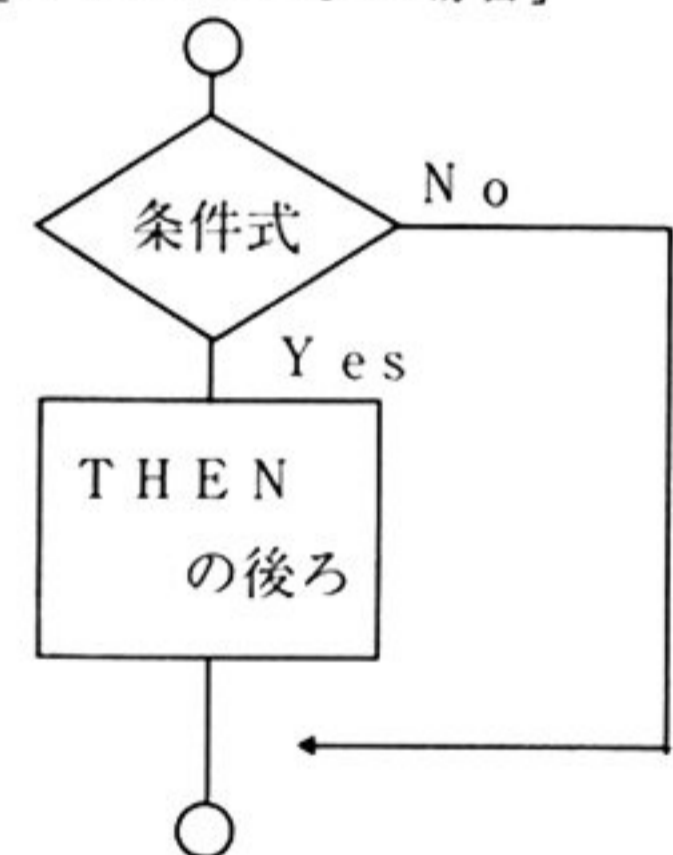
```
IF 条件式 THEN ＊ 〔ELSE ＊〕
```

条件式：関係式、論理式、およびそれらを組み合わせたもの。
＊：（1）行番号。ジャンプ先の行番号。1～65534の整数。
　　（2）ラベル名。LABELで定義された文字列。255文字以内。
　　（3）ステートメント。マルチステートメントも可。

省略形　IF－TH．－EL．

文例　150 IF INKEY$＝" " THEN 300 ELSE 150
⇨スペースキーが押されると行番号300へジャンプし、押されなければこの行の頭に戻ります。

解説　IF－THEN－ELSEは、条件分岐を行なうステートメントです。
IFとTHENの間に条件式を書き、その条件式が成り立つ場合はTHENの後ろを実行し、成り立たない場合はELSEの後ろを実行します。もし、ELSE以降が省略されているときは、次の行に移ります。

［ELSEがある場合］

［ELSEがない場合］

　200 IF X＝10 THEN 500
は、THENをGOTOと置き換えて、
　200 IF X＝10 GOTO 500
と書くことができます。
THENとELSEの後ろに別のIF－THEN－ELSEステートメントを書いてもかまいません。これを入れ子の構造（ネスティング）といいます。
（例）10 IF X＝10 THEN IF Y＝10 THEN
　　200 ELSE 300 ELSE 1000
　　「もし、X＝10ならばTHENの後ろのIF－THEN－ELSEステートメントへ、X≠10ならば1000番へジャンプしなさい」
入れ子は、何重になってもかまいません。

(例) １００ ＩＦ Ｘ＝１０ ＴＨＥＮ ＩＦ Ｙ＝２０ ＴＨＥＮ ＩＦ Ｚ＝３０ ＴＨＥＮ ２００ ＥＬＳＥ ３００ ＥＬＳＥ ４００ ＥＬＳＥ １０００

「Ｘ＝１０かつＹ＝２０かつＺ＝３０ならば、２００番へジャンプしなさい。

Ｘ＝１０かつＹ＝２０かつＺ≠３０ならば３００番へジャンプしなさい。

Ｘ＝１０かつＹ≠２０ならば、４００番へジャンプしなさい。Ｘ≠１０ならば、１０００番へジャンプしなさい」

入れ子の中のＥＬＳＥ文は、省略することができません。最後のＥＬＳＥ文のみ省略することができます。

(例) １００ ＩＦ Ｘ＝１０ ＴＨＥＮ ＩＦ Ｙ＝２０ ＴＨＥＮ ＩＦ Ｚ＝３０ ＴＨＥＮ ２００ ＥＬＳＥ ３００ ＥＬＳＥ ４００

「Ｘ＝１０かつＹ＝２０かつＺ＝３０ならば、２００番へジャンプしなさい。

Ｘ＝１０かつＹ＝２０かつＺ≠３０ならば、３００番へジャンプしなさい。Ｘ＝１０かつＹ≠２０ならば、４００番へジャンプしなさい。Ｘ≠１０ならば、次の行に進みなさい」

ＴＨＥＮおよびＥＬＳＥの後ろは詰めて書くことができますが、前は必ず１文字分のスペースがいります。

(例) １００ ＩＦ Ｘ＝１０ ＴＨＥＮ ＩＦ Ｙ＝２０ ＴＨＥＮ ＩＦ Ｚ＝３０ ＴＨＥＮ２００ ＥＬＳＥ３００ ＥＬＳＥ４００

参照 2.2.14 ＧＯＴＯ

サンプルプログラム

```
10 ' IF-THEN-ELSE (例)
20 PRINT"(Y or N)? ";
30 A$=INPUT$(1)
40 IF A$="Y" THEN 100 ELSE IF A$="N" THEN 200 ELSE 30
100 PRINT:PRINT"Yが押されました。":GOTO20
200 PRINT:PRINT"Nが押されました。":END
RUN
(Y or N)?
Yが押されました。
(Y or N)?
Nが押されました。
Ok
```

３０：キーの入力を待ちます。

４０：押されたキーが[Y]ならば１００へ、[N]なら２００へ、それ以外ならば３０へジャンプします。したがって、[Y]か[N]のキーが押されない限りキーの入力を待ち続けることになります。

2.2.18 FOR−NEXT

機　　能　**FOR−NEXTループ間の処理を繰り返し実行します。**

書　　式

```
FOR i=l TO m 〔STEP s〕
  |
NEXT 〔i〕
```

i：ループ変数。数値型。(単精度型、整数型)
l：初期値
m：終了値。
s：増分。負の数でもよい。省略すると1。

省 略 形

```
F. i=l TO m [STEP s]
  |
N.
```

文　　例

```
FOR I=1 TO 1Ø STEP 2
  |
NEXT
```

⇨I＝1からI＝1Øまで増分2でFOR−NEXT間をループします。

解　　説　FOR−NEXTループにおいて、FORはループの先頭、NEXTはループの終わりを示しています。

たとえば、

```
・・・・・・・・
FOR I=1 TO 1Ø STEP 2
```

繰り返しの処理

```
NEXT I
・・・・・・・
```

という、FOR−NEXTループは次のような動作をします。

プログラムの流れがFORに来ると、ループ変数Iの値が1になって、1回目の処理に入ります。1回目の処理が終わってNEXTを見つけると、Iに増分2が加えられ3となってFORのところまで戻り、2回目の処理に入ります。このようにして、Iの値が3、5、7、9、11と終了値1Øを越えたところでループを終了し、実行がNEXTの次のステートメントに移ります。

また、

```
・・・・・・・
FOR I=10 TO 1 STEP -2
```

繰り返しの処理

```
NEXT I
・・・・・・・
```

のように増分sが負の数のときは、次のような動作をします。

プログラムの流れがＦＯＲに来ると、ループ変数Ｉの値が１０になって、１回目の処理に入ります。１回目の処理が終わってＮＥＸＴを見つけると、Ｉに負の増分－２が加えられ８となってＦＯＲのところまで戻り、２回目の処理に入ります。このようにして、Ｉの値が８、６、４、２、０と終了値１より小さくなったところでループを終了し、実行がＮＥＸＴの次のステートメントに移ります。

なお、「ＳＴＥＰ　ｓ」の部分を省略すると増分は１に設定されます。

ＦＯＲステートメントに対応するＮＥＸＴステートメントがない場合は、「ＦＯＲ　ｗｉｔｈｏｕｔ　ＮＥＸＴ」のエラーが出ます。

ループに入る前から、初期値ｌが終了値ｍをはみ出している場合は、１回もループを通らずに、まっすぐにＮＥＸＴの次のステートメントへ移ります。

ＮＥＸＴの後ろのループ変数ｉは、ＦＯＲのループ変数ｉと同じ変数名でなければなりません。

ＦＯＲ－ＮＥＸＴループの中にさらに別のＦＯＲ－ＮＥＸＴループを入れることができます。これを入れ子の構造（またはネスティング）といいます。

入れ子は何重になってもかまいません。

※　このとき、ループ変数は省略できます。

上の2つはどちらも同じ構造をしています。

ループ変数を整数型にしたり、NEXTの後ろのループ変数を省略すると、FOR－NEXTループのスピードを上げることができます。

なお、次のような構造のプログラムはエラーとなります。

参　照　　2.2.19 REPEAT－UNTIL、2.2.20 WHILE－WEND

サンプルプログラム　　FOR－NEXTの2重の入れ子の構造をもっています。

```
10 ' FOR-NEXT (例)
20 KMODE0
30 SOUND 7,&H3E:SOUND 8,13
40 FOR I=1 TO 20
50 PRINT USING"###";I;
60 FOR J=1 TO I
70 PRINT CHR$(134);
80 NEXT
90 SOUND 0,I*10:SOUND 1,0
100 PRINT
110 NEXT
120 SOUND 7,&H3F
130 KMODE1
```

３０：音を出すための初期設定をします。

４０～１１０：ループ変数をＩとする外側のＦＯＲ－ＮＥＸＴループ。

６０～８０：ループ変数をＪとする内側のＦＯＲ－ＮＥＸＴループ。

９０：音を出すステートメントです。

2.2.19 REPEAT−UNTIL

機能 REPEAT−UNTILループ間の処理を繰り返し実行します。

書式

```
REPEAT
   |
UNTIL 条件式
```

条件式：関係式、論理式、およびそれらを組み合わせた式。

省略形

REP.
|
U.

文例

REPEAT
|
UNTIL X＞10
⇨X＞10になるまでREPEAT−UNTILループをまわります。

解説

REPEAT−UNTILループにおいて、REPEATはループの先頭、UNTILはループの終わりを示します。

たとえば、

```
・・・・・・
REPEAT
┌──────────┐
│繰り返す処理│
└──────────┘
UNTIL X＞10
・・・・・・
```

というREPEAT−UNTILループは次のような動作をします。

プログラムの流れがREPEATのところに来ると、まず1回目の処理に入ります。ループの1回目の処理が終わってUNTILに来ると、変数Xが10より大きいかどうか判断し、大きければ次の行へ、そうでなければREPEATまで戻り、2回目の処理に入ります。このようにして、X＞10が成り立つまでループは回り続けます。したがってREPEATまでプログラムの流れが来ると、必ず1度はループを通ることになります。

なお、REPEAT−UNTILループの中にGOTOステートメントを入れて、ループの外にジャンプすることはおやめください。

参照 2.2.18 FOR−NEXT、2.2.20 WHILE−WEND

サンプル
プログラム

2.2

```
10 ' REPEAT-UNTIL (例)
20 I=0
30 INPUT X
40 REPEAT
50 PRINT I,SQR(I)
60 I=I+1
70 UNTIL I>X
80 PRINT"計算終了"

RUN
? 10
 0          0
 1          1
 2          1.4142136
 3          1.7320508
 4          2
 5          2.236068
 6          2.4494897
 7          2.6457513
 8          2.8284271
 9          3
 10         3.1622777
計算終了
Ok
```

２０：変数Ｉの初期設定を行ないます。

３０：変数Ｘの値を入力します。

４０～７０：ＲＥＰＥＡＴ－ＵＮＴＩＬループ。

５０：変数Ｉの値とその平方根を表示します。

６０：Ｉの値を１増やします。

７０：Ｉの値がＸの値より大きければ次の８０へ抜け出ますが、そうでなければ４０のＲＥＰＥＡＴ－ＵＮＴＩＬループを繰り返します。

2.2.20 WHILE−WEND

機　能　**WHILE−WENDループ間の処理を繰り返し実行します。**

書　式

```
WHILE 条件式
|
WEND
```

条件式：関係式、論理式、およびそれらを組み合わせた式。

省略形

W.
|
WE.

文　例

WHILE　X＞1０
|
WEND
⇨X＞1０が成り立つ間、WHILE−WENDループをまわります。

解　説

WHILE−WENDループにおいて、WHILEはループの先頭、WENDはループの終わりを示しています。

たとえば、

```
・・・・・・
WHILE　X＞10
┌────────┐
│繰り返す処理│
└────────┘
WEND
・・・・・・
```

というWHILE−WENDループは次のような動作をします。

プログラムの流れがWHILEに来ると、Xが1０より大きいかどうか判断し、大きければ1回目の処理に入り、WENDを見つけるとWHILEに戻ります。もし、Xが1０より大きくなければ、ジャンプしてWENDの次の行に移ります。このようにして、X＞1０が成り立たなくなるまでループは回り続けます。したがって、WHILE−WENDループを1度も通らないこともあります。

なお、WHILE−WENDループの中にGOTOステートメントを入れて、ループの外にジャンプすることはおやめください。

参　照　2.2.18 FOR−NEXT、2.2.19 REPEAT−UNTIL

サンプル
プログラム

2.2

```
10 ' WHILE-WEND (例)
20 I=0
30 INPUT X
40 WHILE I<=X
50 PRINT I,SQR(I)
60 I=I+1
70 WEND
80 PRINT"計算終了"
RUN
? 10
 0          0
 1          1
 2          1.4142136
 3          1.7320508
 4          2
 5          2.236068
 6          2.4494897
 7          2.6457513
 8          2.8284271
 9          3
 10         3.1622777
計算終了
Ok
```

２０：変数Ｉの初期設定を行ないます。

３０：変数Ｘの値を入力します。

４０〜７０：ＷＨＩＬＥ－ＷＥＮＤループ。

４０：Ｉの値がＸの値以下ならば次の５０に進みますが、そうでなければＷＥＮＤの次の８０にジャンプします。

５０：Ｉの値とその平方根を表示します。

６０：Ｉの値を１増やします。

７０：４０のＷＨＩＬＥに戻ります。

2.2.21 ON－GOTO／ON－GOSUB

機能　式の値によって、指定された行へ分岐します。

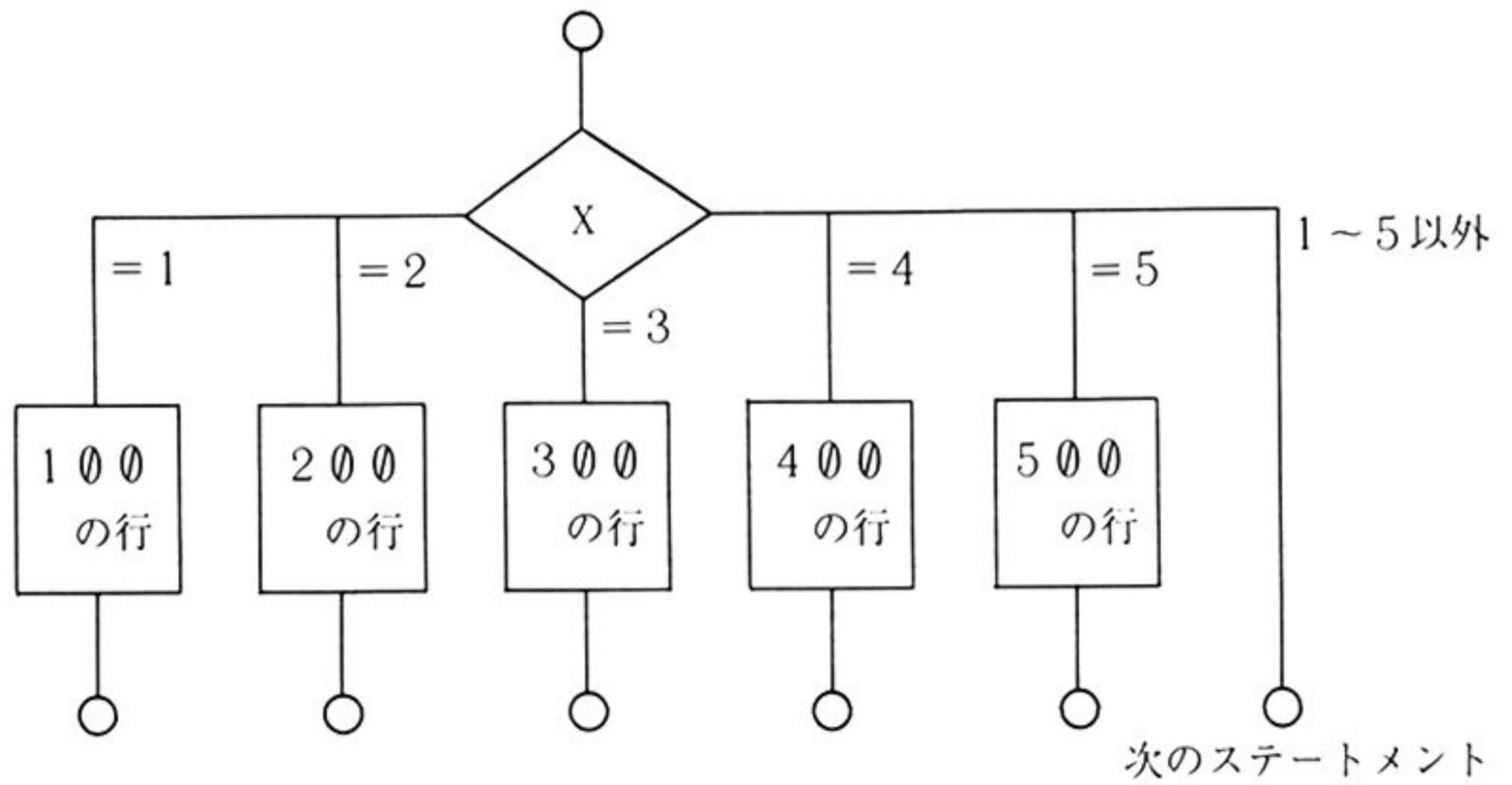
書式

```
〔1〕ON 式 GOTO n1 〔, n2, n3, ……〕
〔2〕ON 式 GOTO L1$ 〔, L2$, L3$, ……〕
〔3〕ON 式 GOSUB n1 〔, n2, n3, ……〕
〔4〕ON 式 GOSUB L1$ 〔, L2$, L3$, ……〕
```

式：数値型の式。
ni：分岐先の行番号。１～６５５３４の整数。i＝１、２、３、………
Ｌｉ＄：分岐先のラベル名。"文字列"。２５５文字以内の文字列。i＝１、２、３、………

文例　ＯＮ　Ｘ　ＧＯＴＯ　１００，２００，３００，４００，５００
⇨Ｘの値が１のとき行番号１００へ、２のとき行番号２００へ、３のとき行番号３００へ、４のとき行番号４００へ、５のとき行番号５００へジャンプします。Ｘの値がそれ以外であれば次のステートメントが実行されます。

解説　式の値がｉのとき、ｉ番目の行番号（ラベル名）の行にジャンプします。
すなわち、式の値が１ならば、１番目の行番号 n_1 またはラベル名 L_1＄へ、
式の値が２ならば、２番目の行番号 n_2 またはラベル名 L_2＄へ、
式の値が３ならば、３番目の行番号 n_3 またはラベル名 L_3＄へ、
・・・・・・・・・・・・・・・・・・・・・・・・・・・
というように、分岐先へジャンプします。

（例）ＯＮ　Ｘ　ＧＯＴＯ　１００，２００，３００，４００，５００
||
ＩＦ　Ｘ＝１　ＴＨＥＮ　１００
ＩＦ　Ｘ＝２　ＴＨＥＮ　２００
ＩＦ　Ｘ＝３　ＴＨＥＮ　３００
ＩＦ　Ｘ＝４　ＴＨＥＮ　４００
ＩＦ　Ｘ＝５　ＴＨＥＮ　５００

式の値が負、0、あるいは指定した行番号やラベル名の数より大きいときは、次のステートメントが実行されます。

式の値が整数でないときは、小数第1位で四捨五入された値に変換されます。

ＯＮ―ＧＯＳＵＢステートメントにおいて、分岐先の行番号（ラベル名）はサブルーチンの先頭でなければなりません。このとき、サブルーチンを実行しＲＥＴＵＲＮに達すると、ＯＮ－ＧＯＳＵＢの次のステートメントに戻ってきます。

参　照　　2.2.14 ＧＯＴＯ、2.2.15 ＧＯＳＵＢ

サンプルプログラム　　占いのひな型といえるプログラムを示します。

```
10 ' ON-GOSUB (例)
20 INPUT" 0 から 5 のうち何番が好き";N
30 ON N+1 GOSUB 100,200,300,400,500,600
40 PRINT N;"は「";X$;"」の象徴です。"
50 PRINT
60 GOTO 20
100 X$="永遠":RETURN
200 X$="希望":RETURN
300 X$="女性":RETURN
400 X$="男性":RETURN
500 X$="完全":RETURN
600 X$="結婚":RETURN
```

２０：Ｎの値を入力します。

３０：Ｎ＋１の値によって１００～６００のサブルーチンを呼び出します。

４０：メッセージを表示します。５０は改行のためのＰＲＩＮＴです。

６０：２０に戻ります。

2.2.22 ON−RETURN／ON−RESUME

機能 式の値によって、指定された行へ復帰します。

書式

```
〔1〕ON 式 RETURN n1 〔, n2, n3, ……〕
〔2〕ON 式 RETURN L1$ 〔, L2$, L3$, ……〕
〔3〕ON 式 RESUME n1 〔, n2, n3, ……〕
〔4〕ON 式 RESUME L1$ 〔, L2$, L3$, ……〕
```

式：数値型の式。

ni：復帰先の行番号。1～65534の整数。i＝1、2、3、………

Li$：復帰先のラベル名。255文字以内の文字列。i＝1、2、3、………

文例 ON X RETURN 100, 200, 300, 400, 500

⇨Xの値が1のとき行番号100へ、2のとき行番号200へ、3のとき行番号300へ、4のとき行番号400へ、5のとき行番号500へ復帰します。Xの値がそれ以外であれば次のステートメントが実行されます。

解説 このステートメントは、サブルーチンやエラー処理ルーチンの最後に書いて、そこからメインプログラムに戻るときの、戻り先を指定します。

式の値がiのとき、i番目の行番号またはラベル名の行に戻ります。

すなわち、式の値が1ならば、1番目の行番号 n_1 またはラベル名 $L_1$$へ、
式の値が2ならば、2番目の行番号 n_2 またはラベル名 $L_2$$へ、
式の値が3ならば、3番目の行番号 n_3 またはラベル名 $L_3$$へ、
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・

というように、復帰先へジャンプします。

(例) ON X RETURN 100, 200, 300, 400, 500

‖

```
IF X=1 THEN RETURN 100
IF X=2 THEN RETURN 200
IF X=3 THEN RETURN 300
IF X=4 THEN RETURN 400
IF X=5 THEN RETURN 500
```

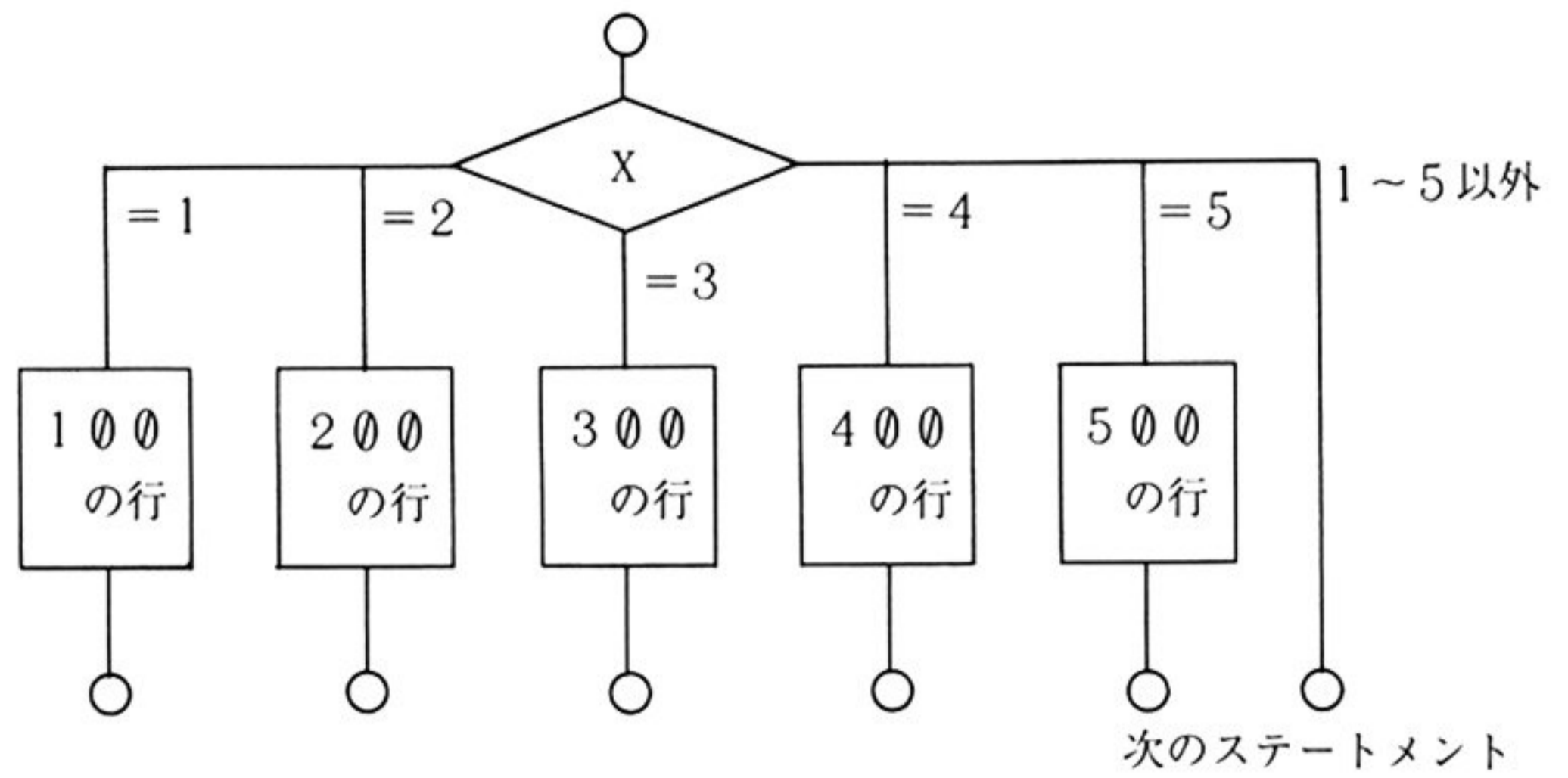
式の値が負、０、あるいは指定した行番号やラベル名の数より大きい数より大きいときは、次のステートメントが実行されます。

参　照　2.2.21 ON－GOSUB、2.2.16 RETURN、2.5.2 RESUME

2.2.23 ON−RESTORE

機　能　**式の値によって、読み込むデータ文を変更します。**

書　式

〔1〕**ON　式　RESTORE　n_1　〔, n_2, n_3, ……〕**
〔2〕**ON　式　RESTORE　$L_1$$　〔, $L_2$$, $L_3$$, ……〕**

式：数値型の式。
ni：読み始めるデータ文の行番号。１〜６５５３４の整数。i＝１、２、３、……
Ｌｉ＄：読み始めるデータ文のラベル名。２５５文字以内の文字列。i＝１、２、３、……

文　例

ＯＮ　Ｘ　ＲＥＳＴＯＲＥ　１０００，２０００，３０００
⇨ＲＥＡＤ文で読みこむデータの先頭行を、Ｘの値が１のとき行番号１０００、２のとき行番号２０００、３のとき行番号３０００に設定します。

解　説

このステートメントは、式の値によって読み込むデータ文（ＤＡＴＡ文）を変更するためのものです。
式の値がｉのとき、ｉ番目の行番号（ラベル名）の行を指定します。
すなわち、式の値が１ならば、１番目の行番号 n_1 またはラベル名 L_1＄を、
式の値が２ならば、２番目の行番号 n_2 またはラベル名 L_2＄を、
式の値が３ならば、３番目の行番号 n_3 またはラベル名 L_2＄を、
・・・・・・・・・・・・・・・・・・・・・・・・・・・・
というように、ＲＥＡＤステートメントで読み込む先頭のＤＡＴＡ文を指定します。

```
(例) 100  ON X RESTORE 1000, 2000, 3000
            ||
     100  ON X GOSUB 110, 120, 130
     105  GOTO150
     110  RESTORE 1000:RETURN
     120  RESTORE 2000:RETURN
     130  RESTORE 3000:RETURN
     150  ・・・・・・・・・・・・・
```

式の値が負、０、あるいは指定した行番号やラベル名の数より大きいときは、次のステートメントが実行されます。

参　照

2.3.5 ＲＥＳＴＯＲＥ

サンプル
プログラム

2.2

```
10 ' ON-RESTORE (例)
20 S=0
30 INPUT"サンプルの選択(1-4)";X
40 ON X RESTORE1000,2000,3000
50 IF X=4 THEN END
60 READ N
70 PRINT"サンプル数=";N
80 PRINT"サンプルデータ"
90 FOR I=1 TO N
100 READ D
110 PRINT D;
120 S=S+D
130 NEXT
140 A=S/N
150 PRINT:PRINT"平均値     =";A
160 PRINT:GOTO30
1000 DATA 10
1010 DATA 23,22,25,21,23,22,24,22,21,23
2000 DATA 7
2010 DATA 21,21,22,23,24,23,23
3000 DATA 9
3010 DATA 23,22,22,22,21,22,23,23,23
```

```
RUN
サンプルの選択(1-4)? 1
サンプル数= 10
サンプルデータ
 23  22  25  21  23  22  24  22  21  23
平均値      = 22.6

サンプルの選択(1-4)? 2
サンプル数= 7
サンプルデータ
 21  21  22  23  24  23  23
平均値      = 54.714286

サンプルの選択(1-4)? 3
サンプル数= 9
サンプルデータ
 23  22  22  22  21  22  23  23  23
平均値      = 64.888889

サンプルの選択(1-4)? 4
Ok
```

３０：Ｘの値の入力をします。

４０：Ｘの値によってデータ文の先頭を１００，２０００，３０００のいずれかに設定します。

６０：ＤＡＴＡ文からサンプル数を読み込んでＮに代入します。

８０～１２０：データをＮ個読み込んでその和を求めるＦＯＲ－ＮＥＸＴループ。

１３０：データの和Ｓをサンプル数Ｎで割って平均値Ａを求めます。

１４０：平均値Ａを表示します。

１０００～３０１０：データ文

2.2.24 DEF FN

機　　能　**ユーザーの書いた関数を登録します。**

書　　式

〔1〕**DEF　FNname（仮引数）＝式**
〔2〕**DEF　FNname（仮引数，仮引数，……）＝式**

ｎａｍｅ：ＦＮに続いて書かれる変数名。数値変数、文字変数のどちらも可能。
仮引数：右辺の式の中にある変数名。
式：ユーザの書く関数。型はｎａｍｅに一致していなければならない。

文　　例　ＤＥＦ　ＦＮＳＥＣ（Ｘ）＝１／ＣＯＳ（Ｘ）
⇨セカント関数ＳＥＣ（Ｘ）を１／ＣＯＳ（Ｘ）と定義します。

解　　説　ＢＡＳＩＣに用意されていない関数を使いたいときは、自分で作らなければなりません。そのための一般的な方法は、サブルーチンを作ることですが、関数の方が１行で書ける場合は、この「ＤＥＦ　ＦＮステートメント」を使って関数を定義します。

ｎａｍｅは、関数の名前で、得たい値が数値であれば数値型の変数名、文字列であれば文字型の変数名をつけます。

参　　照　2.2.25 ＤＥＦ　ＵＳＲ、3.8.7 ＦＮ

サンプルプログラム

```
10 ' DEF FN (例)
20 DEF FNA(X,Y,Z)=(X+Y+Z)/3
30 DEF FNB(X,Y,Z)=SQR((X^2+Y^2+Z^2)/3)
40 DEF FNC(X,Y,Z)=(X*Y*Z)^(1/3)
50 INPUT"X=";X
60 INPUT"Y=";Y
70 INPUT"Z=";Z
80 FA=FNA(X,Y,Z)
90 FB=FNB(X,Y,Z)
100 FC=FNC(X,Y,Z)
110 PRINT"相加平均=";FA
120 PRINT"平方平均=";FB
130 PRINT"相乗平均=";FC
```

```
RUN
X=? 3
Y=? 5
Z=? 7
相加平均= 5
平方平均= 5.2599113
相乗平均= 4.717694
Ok
```

２０：関数ＦＮＡ（Ｘ，Ｙ，Ｚ）を $\frac{X+Y+Z}{3}$ と定義します。
３０：関数ＦＮＢ（Ｘ，Ｙ，Ｚ）を $\sqrt{\frac{X^2+Y^2+Z^2}{3}}$ と定義します。
４０：関数ＦＮＣ（Ｘ，Ｙ，Ｚ）を $\sqrt[3]{XYZ}$ と定義します。
５０～７０：変数Ｘ，Ｙ，Ｚの値の入力をします。
８０～１００：３つの関数の値をそれぞれＦＡ、ＦＢ、ＦＣに代入します。
１１０～１３０：ＦＡ、ＦＢ、ＦＣの値を表示します。

2.2

2.2.25 DEF USR

機能

機械語のユーザー関数を指定します。

書式

```
DEF USR〔n〕＝a
```

n：ユーザー関数識別番号。0～9の整数。
a：機械語のユーザー関数の開始アドレス。

文例

DEF USR0＝&HE000
⇨機械語サブルーチン0の実行開始アドレスを&HE000に設定します。

解説

「DEF USR」は、ユーザー関数（USR）が呼び出す、機械語サブルーチンの開始アドレスを指定するためのステートメントです。

「DEF USR」の後ろの番号nは、0～9までの整数で表わされ、USR関数の識別番号となります。nは何度でも指定しなおすことができます。

機械語サブルーチンの使い方については、『ユーザーズマニュアル』の「機械語サブルーチンとモニタ」を参照してください。

参照

『ユーザーズマニュアル』の「機械語サブルーチンとモニタ」、2.2.26 CALL、3.8.8 USR

サンプルプログラム

メモリのE000番地から簡単な機械語プログラムを作り、DEF USRで定義したUSR関数を呼び出してみましょう。

```
10 ' DEF USR (例)
20 CLEAR &HE000
30 I=0
40 READ D$
50 D=VAL("&H"+D$)
60 POKE &HE000+I,D
70 IF D=&HC9 THEN 100
80 I=I+1
90 GOTO 40
100 DEF USR0=&HE000
110 DEF USR1=&HE100
120 A=10
130 X=USR0(A)
140 END
150 '
160 DATA ED,43,00,E2 :'LD  (0E200H),BC
170 DATA ED,53,02,E2 :'LD  (0E202H),DE
180 DATA 22,04,E2    :'LD  (0E204H),HL
190 DATA ED,73,06,E2 :'LD  (0E206H),SP
200 DATA DD,22,08,E2 :'LD  (0E208H),IX
210 DATA FD,22,0A,E2 :'LD  (0E20AH),IY
220 DATA 32,0C,E2    :'LD  (0E20CH),A
230 DATA 06,00       :'LD  B,0
240 DATA 4F          :'LD  C,A
250 DATA 11,0D,E2    :'LD  DE,0E20DH
260 DATA ED,B0       :'LDIR
270 DATA C9          :'RET
```

２０：ＤＦＦＦ番地までをＢＡＳＩＣで使用するエリアに設定します。

３０～９０：ＤＡＴＡ文で用意された機械語プログラムをメモリのＥ０００番地以降に書き込みます。

１００，１１０：ＤＥＦ ＵＳＲ文によってＵＳＲ関数０の呼び出すアドレスをＥ０００番地、ＵＳＲ関数１の呼び出すアドレスをＥ１００番地に指定します。

１３０：ＵＳＲ関数０を呼び出します。

１６０～２７０：ＣＰＵの各レジスタの内容をメモリのＥ２００番地以降に書き込む機械語サブルーチンを表すデータです。


```
RUN
Ok
MON
*D E200 E21F
:E200=00 E0 3D 41 00 DF D8 DE /.※A.*ﾘﾞ
:E208=11 02 7D 72 05 84 20 00 /..}r.※.
:E210=00 00 58 30 47 30 34 30 /..X0G040
:E218=29 30 1E 30 13 30 08 30 /)0.0.0.0
*R
Ok
```

2.2.26 CALL

2.2

機能

機械語サブルーチンを呼び出します。

書式

```
CALL a
```

a：機械語サブルーチンの開始アドレス。

省略形

CA.

文例

CALL &HE000
⇨実行開始アドレスが&HE000の機械語サブルーチンを呼び出します。

解説

機械語サブルーチンは、CALLステートメントによって、プログラムから呼び出すことができます。

aは、機械語サブルーチンの開始アドレスです。

機械語サブルーチンからBASICプログラムへはRET命令（$C9）で戻れます。

機械語サブルーチンについて、詳しくは、『ユーザーズマニュアル』の「機械語サブルーチンとモニタ」を参照してください。

参照

『ユーザーズマニュアル』の「機械語サブルーチンとモニタ」、
2.2.25 DEF USR

2.2.27 RANDOMIZE

機能　**RND関数で発生させる乱数の初期値を設定します。**

書式

```
RANDOMIZE〔x〕
```

x：数式。－32768～65535の整数を値にもつこと。

省略形　RA.

文例　RANDOMIZE VAL (RIGHT$ (TIME$, 2))

⇨タイマー変数の秒数によって、RND関数で発生する乱数の初期値を設定します。

解説　RND関数は、プログラムを実行するたびに違う系列の乱数を発生しますが、このステートメントでxの値を指定することによって、乱数の初期値を設定することができます。

xの値によって乱数の初期値が決定されます。

xの値を省略すると初期値は定まらなくなります。

参照　3.1.20 RND

2.2.28　ＰＵＳＨ

2.2

機　能　　データをユーザースタックに書き込みます。

書　式

```
ＰＵＳＨ　式〔，式，………〕
```

式：数値、文字列を表わす式。単独の定数、変数、配列を含む。

文　例　　ＰＵＳＨ　Ａ，Ａ＄

⇨ユーザースタックに変数Ａと文字変数Ａ＄の値を書き込みます。

解　説　　数式や文字式などのデータをユーザースタックに順に書き込みます。このステートメントは式の値を一時保持しておきたい場合などに有効となります。つまり、データを一時退避させておくステートメントです。また、１つの変数で、配列のかわりに扱えます。ただし、データのユーザースタックからの読み出しは、ＰＯＰステートメントで行ないます。

〔ユーザースタックについて〕

ＢＡＳＩＣの変数エリア後に確保される、ＰＵＳＨ、ＰＯＰステートメント用のワークエリアで、ＦＩＬＯ（ファースト　イン　ラスト　アウト）の構造のものです。

（「ＦＩＬＯ」とは、Ａ、Ｂ、Ｃの順に入っているものが、読み出す時には、最後に入ったものからＣ、Ｂ、Ａの順に読み出されます。）

ですから、ＰＵＳＨされたデータが、文字列や数値の混在したものであった場合、ＰＯＰする時は、順序を間違えないように注意してください。「Ｔｙｐｅ　ｍｉｓｍａｔｃｈ」エラーとなります。

参　照　　2.2.29 ＰＯＰ

サンプルプログラム

変数AのデータをＰＵＳＨし、その後ＰＯＰした場合、書き込まれる順序と読み出された順序が違っています。

```
10 ' PUSH / POP (例)
20 INIT:WIDTH 80,25
30 DIM A(20)
40 PRINT "PUSH  A"
50 FOR I=0 TO 50
60 A=I
70 PUSH A
80 PRINT A;
90 NEXT
100 PRINT
110 PRINT "POP    A    B$"
120 FOR J=0 TO 20
130 POP A
140 PRINT A;
150 B$=CHR$(A)
160 PRINT #0,B$;
170 NEXT
180 END

RUN
PUSH  A
 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14
 15  16  17  18  19  20  21  22  23  24  25  26  2
7  28  29  30  31  32  33  34  35  36  37  38  39
 40  41  42  43  44  45  46  47  48  49  50
POP    A    B$
 50 2 49 1 48 0 47 / 46 . 45 - 44 , 43 + 42 * 41 )
 40 ( 39 ' 38 & 37 % 36 $ 35 # 34 " 33 ! 32   31 ↓
 30 ↑
Ok
```

2.2.29 ＰＯＰ

機　能

ユーザースタックから順にデータを読み出します。

書　式

ＰＯＰ〔変数〕〔，変数，………〕

変数：数値変数、文字変数または配列変数。

文　例

ＰＯＰ　Ｂ＄，Ｂ

⇨ユーザースタックから文字変数Ｂ＄と変数Ｂにデータを返します。

解　説

ＰＵＳＨステートメントによってユーザースタックに書き込まれたデータを読み出して変数に代入します。

ＰＯＰのみを実行すると、ユーザースタックの初期化をします。

注意：ＰＵＳＨステートメントによって書き込まれたデータは、ＰＯＰすると、最後に書き込まれたデータから読み出されますので、注意してください。

参　照

2.2.28 ＰＵＳＨ

2.2.30 STOP ON／STOP OFF

機能 ブレークキーの機能の許可、禁止を設定します。

書式

```
〔1〕STOP ON
〔2〕STOP OFF
```

文例 ＳＴＯＰ　ＯＦＦ

⇨ブレークキーがなくなります。

解説 ⇨ブレークキー（SHIFT＋BREAKキー）の機能を有効にしたり、無効にしたりします。

（例）ＳＴＯＰ　ＯＮ……ブレークキーの機能を許可し有効にします。以後SHIFT＋BREAKキーが押されると、プログラムの実行を停止します。

ＢＡＳＩＣ起動時は、この状態です。

（例）ＳＴＯＰ　ＯＦＦ…ブレークキーの機能を禁止します。以後SHIFT＋BREAKキーを押してもプログラムを停止することはできません。ＩＰＬリセットスイッチを押してください。

ただし、ＳＡＶＥコマンドのＰオプションのセーブと同じく、プログラム中にＥＮＤ・ＳＴＯＰステートメントがあると終了・停止します。

参照 2.1.20 ＳＡＶＥ

2.3

2.3 入出力ステートメント

2.3.1 INPUT

機能　キーボードからのデータを入力します。

書式

```
INPUT〔"メッセージ";〕変数〔,変数,……〕
```

"メッセージ"：画面に表示されるメッセージ。255文字以内の文字列。普通は、入力の要求メッセージが書かれます。この後のセミコロン（;）は、カンマ（,）と置き換えることができます。

変数：キーボードから入力されたデータの入る変数。

省略形　I.

文例　INPUT" **データは** ";X

⇨キーボードから打った値を変数Xに入力します。画面には、「 **データは** ?」と表示されます。

解説　キーボードから入力した数値や文字のデータを変数に入れます。変数は数値変数、文字変数いずれの型でもかまいませんが、入力するデータの型と一致していなければなりません。

(例) 10　INPUT　X……実行すると、「?」が表示されるので、変数Xの値をキーボードから入れて⏎キーを押す。

変数は、カンマ（,）で区切って複数個書くことができます。この場合、データを入力するときに、対応するデータをカンマ（,）で区切って1度に入力します。画面では右端に来ると自動的に次の行に移りますが、それにかまわず入力します。

(例) 10　INPUT　X, Y, Z……実行すると、「?」が表示されるので、X, Y, Zの3個の値をカンマ（,）で区切って入れて⏎キーを押します。

文字変数には、ほとんどどんな文字列でも入力することができますが、カンマ（,）が含まれる文字列は、カンマが区切り記号として使われるため、引用符（"）で囲まなければ入力することができません。

INPUTステートメントを実行すると、画面に「?」を表示して入力待ちになりますが「?」だけでは何を入力するのかわかりません。そこで、INPUTにはメッセージの表示機能がついていて、画面に入力の要求メッセージを出すことができます。

(例) 10　INPUT" **データは** ";X……実行すると、

「 **データは** ?」と表示されるので、Xの値を入れて⏎キーを押します。

「？」を出さないようにするには、メッセージの次にセミコロン（;）の代わりにカンマ（,）を入れます。

(例) １０　ＩＮＰＵＴ"**データは**　"，Ｘ……実行すると、

「**データは**」と表示されるので、Ｘの値を入れて⏎キーを押します。

ＩＮＰＵＴステートメントに書かれた変数の数と、入力したデータの数とが合わない場合、入力したデータの数の方が多いときは、余分なデータが無視され、入力したデータの数の方が少ないときは、後の変数の値が変わらないで保存されます。

何も入力しないで⏎キーを押すと、変数の値は現在のまま保存されます。

2.3

参　照　　2.3.3 ＲＥＡＤ、2.4.7 ＩＮＰＵＴ#、3.3.6 ＩＮＰＵＴ$

サンプルプログラム

```
10 ' INPUT (例)
20 INPUT"A=";A
30 INPUT"B=";B
40 C=A+B
50 D=A-B
60 E=A*B
70 F=A/B
80 PRINT"A+B=";C
90 PRINT"A-B=";D
100 PRINT"A*B=";E
110 PRINT"A/B=";F
RUN
A=? 5
B=? 3
A+B= 8
A-B= 2
A*B= 15
A/B= 1.6666667
Ok
```

Ａ＝　，Ｂ＝　という問いに答えると、Ａ，Ｂ，Ａ＋Ｂ，Ａ－Ｂ，Ａ/Ｂを計算し表示するプログラムです。

2.3.2 LINPUT/LINE INPUT

機　　能　キーボードから文字を入力します。

書　　式

```
〔1〕LINPUT〔"メッセージ";〕x$
〔2〕LINE　INPUT〔"メッセージ";〕x$
```

"メッセージ"：画面に表示されるメッセージ。２５５文字以内の文字列。普通は、入力の要求メッセージが書かれます。この後のセミコロン（;）は、カンマ（,）と置き換えることができます。

x＄：キーボードから入力された文字データの入る変数。

省 略 形　〔1〕LIN.
〔2〕LINEI.

文　　例　LINPUT　x＄
⇨1行分をそのまま文字変数x＄に入力します。

解　　説　キーボードから入力した文字のデータを1つの文字変数に入れます。

入力するデータは２５５文字以内でなければなりません。画面では右端に来ると自動的に次の行に移りますが、それにかまわず入力します。2、3行となってもかまいません。

（例）１０　LINPUT　X＄……実行すると、入力待ちになるので、X＄に入る文字列をキーボードから打ち込んで⏎キーを押します。

文字変数には、ほとんどどんな文字列でも入力することができます。INPUTでそのまま入力できなかったカンマ（,）も入力することができます。

INPUTステートメントのように、画面に「？」は表示されません。

LINPUTおよびLINE　INPUTにはメッセージの表示機能がついていて、画面に入力の要求メッセージを出すことができます。ただし、このときメッセージもデータとして変数に入ってしまいます。入力した文字列だけをデータにしたい場合は、プログラム中にメッセージの部分を取り除く処理を加える必要があります。

（例）１０　LINPUT"住所：";AD＄……実行すると、「住所：」と表示されるので、AD＄の文字列を入れて⏎キーを押します。

参　　照　2.3.1 INPUT、2.4.8 LINPUT#

サンプル
プログラム

```
10 ' LINPUT (例)
20 ON ERROR GOTO 80
30 LINPUT A$
40 IF A$="" THEN 30
50 IF A$="END" THEN END
60 PRINT CALC(A$)
70 GOTO 30
80 PRINT"計算できません。"
90 RESUME 30
RUN
20*3+40
 100
END
Ok
```

30行目の入力待ちに、数式（たとえば20×3+40）を入力すると、計算結果を表示するプログラムです。

2.3.3 READ

機　能　DATAステートメントで用意されたデータを入力します。

書　式

```
READ 変数〔,変数,……〕
```

変数：DATAステートメントのデータの入る変数。数値変数または文字変数。

文　例　READ X, Y, Z

⇨DATAステートメントに用意された数値データを読み込んで、数値変数X、Y、Zにそれぞれの値を代入します。

解　説　データの入力には、INPUTステートメントによる方法とREADステートメントによる方法があります。

INPUTステートメントを使うと、プログラムの実行時にキーボードからデータを入力することができますが、実行のたびに同じデータを入力したり、多量のデータを入力するときはたいへんです。このようなときには、プログラム中にDATAステートメントであらかじめデータを入れておいて、それをREADステートメントで読み込むようにします。そうすれば同じデータを何度でも利用することができます。

READは、DATAステートメントと対にして使うステートメントで、READの後ろに変数を書き、それに対する定数データをDATAステートメントに書きます。このとき、READの変数の型とDATAの定数データの型は、一致していなければなりません。

DATAステートメントは、実行にはかかわらないので、プログラム中のどこに置いてもよく、1つのREADステートメントで2つ以上のDATAステートメントからデータを読み込んだり、RESTOREステートメントを使っていくつかのREADステートメントで1つのDATA文を読み込むことができます。どちらの場合も、DATAステートメントは行番号の若い順に、データの並びの左側から読み込まれます。

READで読み込むデータの数がDATAステートメントのデータの数より多い場合は、「Out of data」のエラーが出ます。反対に、READで読み込むデータの数がDATA文のデータの数より少ない場合は、残ったデータが無視されます。ただし、次にREADステートメントがあるときは、残ったデータの先頭から引き続き読み込まれます。

RESTOREステートメントを使うと、読み込むDATAステートメントの行を変更したり、同じDATAステートメントを何度も読み込むことができます。

参　照　2.3.4 DATA、2.3.5 RESTORE

サンプル
プログラム

```
10 ' READ (例)
20 FOR I=1 TO 10
30 READ X
40 PRINT X;
50 NEXT
60 END
70 DATA 21,35,54,16,77,43,95,88,61,0
RUN
 21  35  54  16  77  43  95  88  61  0
Ok
```

2.3

７０行目のデータを３０行目で変数Xに読み込み表示させるプログラムです。

2.3.4 DATA

2.3

機能 **READで読み込むデータを用意します。**

文例

DATA 定数〔,定数,……〕

定数：数値定数、"文字列"（文字定数)。

省略形 DA.

文例 DATA 23, 44, 5Ø, 33, 89, 8Ø

⇨READステートメントで読み込む数値データを23, 44, 5Ø, 33, 89, 8Øの6つ用意します。

DATA Makiko, Yayoi, Noriko

⇨READステートメントで読み込む文字型データをMakiko, Yayoi, Norikoの3つ用意します。

解説 DATAは、READステートメントで読み込むデータを用意するステートメントです。DATAステートメントに書けるデータは、数値定数および文字定数（文字列）で、1つのDATAステートメントにつき255文字分のデータを書くことができます。

実行にはかかわらないので、プログラム中のどこにでも置くことができますが、プログラムの読みやすさという点から、READステートメントのすぐ下の行に書くか、プログラムの最後にまとめて書くのが一般的です。

文字列をデータとして用意するとき引用符（"）で囲む必要はありません。ただし、文字列の中にカンマ(,)やコロン（：）の区切り記号を含む場合は、引用符で囲む必要があります。

参照 2.3.3 READ、2.3.5 RESTORE

サンプルプログラム

それぞれのデータを読み込みそのデータを表示します。

（例１）
```
10 ' DATA (例1)
20 READ D
40 IF D=-1 THEN END
45 PRINT D,
50 GOTO20
60 DATA 21,35,54,16,77,43,95,88,61,23
70 DATA 52,54,78,54,56,27,33,54,64,87
80 DATA 36,83,23,84,13,72,66,77,36,91
90 DATA -1
RUN
 21        35        54        16
 77        43        95        88
 61        23        52        54
 78        54        56        27
 33        54        64        87
 36        83        23        84
 13        72
Ok
```

６０行～９０行のＤＡＴＡを数値データとして読み込み、読み込んだ値が－１でなければ表示し、－１であれば終了します。

（例２）
```
10 ' DATA (例2)
20 READ D$
30 IF D$="END" THEN END
40 D=VAL(D$)
50 PRINT D,
60 GOTO20
70 DATA 21,35,54,16,77,43,95,88,61,23
80 DATA 52,54,78,54,56,27,33,54,64,87
90 DATA 36,83,23,84,13,72,66,77,36,91
100 DATA END
RUN
 21        35        54        16
 77        43        95        88
 61        23        52        54
 78        54        56        27
 33        54        64        87
 36        83        23        84
 13        72
Ok
```

（例１）と同じですがデータを文字データとして読み込み、これが"ＥＮＤ"と同じであるか否か判別し、"ＥＮＤ"でないとき、数値に変換し表示し"ＥＮＤ"の時終了します。

（例３）
```
10 ' DATA (例3)
20 READ D$
30 IF D$="END" THEN END
50 PRINT D$,
60 GOTO20
70 DATA Atsuko,Fumiko,Yoshio,Megumi
80 DATA Takako,Yuuji,Madoka,Mariko
90 DATA Yoshinori,Junko,Naomi,Hisami
100 DATA END
RUN
Atsuko    Fumiko    Yoshio    Megumi
Takako    Yuuji     Madoka    Mariko
Yoshinori Junko     Naomi     Hisami
Ok
```

文字データを文字データとして読み込み表示します。

2.3.5 RESTORE

2.3

機　　能　**READステートメントで読み込むDATAステートメントを変更します。**

書　　式

```
〔1〕RESTORE　〔n〕
〔2〕RESTORE　〔L$〕
```

n：行番号。読み始めるDATAステートメントの先頭。
L$：ラベル名。"文字列"。読み始めるDATAステートメントの先頭。
LABELで定義した文字列。

省 略 形　RES.

文　　例　RESTORE　1000
⇨READステートメントで読み始めるDATAステートメントの行番号を1000とします。

解　　説　BASICは、DATAステートメントを指すポインタを持っています。READステートメントを実行すると、プログラムの先頭からDATAステートメントを探して行き、最初に現われたDATAステートメントの行番号がポインタにセットされます。

RESTOREでDATAステートメントの行を指定すると、ポインタがそこにセットされますから、READステートメントの直前にRESTORE命令で次のREAD命令で読み込みたいDATAの格納されている行番号、ラベルを指定しておけば、同じDATAステートメントを何度も利用したり、読み込みたいDATAステートメントを任意に指定することができます。

RESTOREのみを指定すると、ポインタがプログラムの先頭にセットされます。

なお、このポインタの値は、システム変数のDTLに入っています。

参　　照　2.3.3 READ、2.3.4 DATA、3.10.3 DTL、
2.2.23 ON RESTORE

サンプル
プログラム

ＲＥＳＴＯＲＥステートメントで指定された行番号から、ＤＡＴＡを読み込み表示します。

```
10 ' RESTORE (例)
20 RESTORE2000
30 FOR I=0 TO 5
40 READ A
50 PRINT A;
60 NEXT
70 PRINT
80 RESTORE"入力データ"
90 FOR I=0 TO 5
100 READ A
110 PRINT A;
120 NEXT
130 END
1000 LABEL"入力データ"
1010 DATA 23,43,55,65,42,93
2000 DATA 12,56,34,68,53,22
RUN
 12  56  34  68  53  22
 23  43  55  65  42  93
Ok
```

１０００行目及び１０１０行目のデータと２０００行目のデータを、ＲＥＳＴＯＲＥステートメントをつかって読み込む順番を制御しています。

2.3

2.3.6 PRINT

2.3

機　能　　画面にデータを表示します。

書　式

```
PRINT 式〔, 式, ……〕
```

式：数値、文字式。単独の定数、変数、配列も含みます。
　　カンマ（,）セミコロン（;）と置き換えることができます。

省略形　　P．または　？

文　例　　PRINT　X
⇨Xの値を表示します。

解　説

　PRINTは、画面上に、定数の値、数値変数の値、文字変数の値（文字列）、および式の値を表示します。

　PRINTは、「印刷する」という意味ですが、これは初期の出力装置のほとんどがテレタイプであったことに由来します。

（例1）PRINT　A⏎
　　⇨変数Aの値を表示します。

（例2）PRINT　"A"⏎
　　⇨文字Aを表示します。

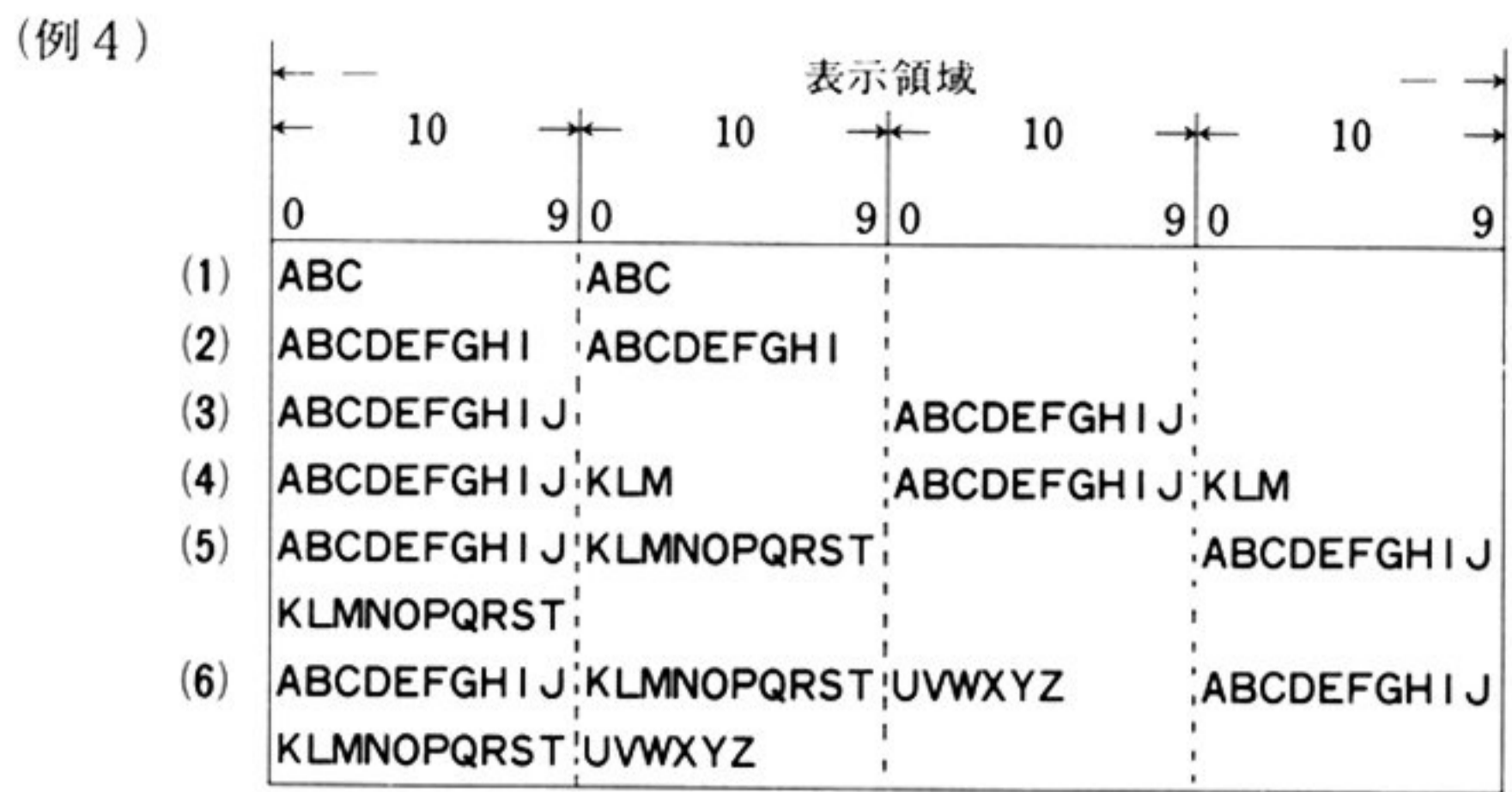
　PRINTによって画面に表示される値は、カンマ（,）かセミコロン（;）で区切ることによって、複数個並べて書くことができます。（このとき、カンマやセミコロンを区切り記号「セパレータ」といいます。）

　複数の式をセミコロン（;）で区切って書くと、それぞれの値をその前の値のすぐ後ろに表示することができます。文字列のときは、ぴったりくっついて表示されますが、数値のときは、後ろに必ず空白が1つ、正の数の前にも空白が1つ置かれます。

（例3）PRINT"ABC";"DEF"⏎
```
ABCDEF
Ok
```

　複数の式を「カンマ」で区切って書くと、10文字単位で画面上に並んで表示されます。40字モードのとき、画面の表示領域は次のように10文字単位で4つのブロックに分割されていて、表示される値は必ず各ブロックの初めから表示されます。

（例４）

	表示領域			
	←10→	←10→	←10→	←10→
	0　　　　　9	0　　　　　9	0　　　　　9	0　　　　　9
(1)	ABC	ABC		
(2)	ABCDEFGHI	ABCDEFGHI		
(3)	ABCDEFGHIJ		ABCDEFGHIJ	
(4)	ABCDEFGHIJ	KLM	ABCDEFGHIJ	KLM
(5)	ABCDEFGHIJ	KLMNOPQRST		ABCDEFGHIJ
	KLMNOPQRST			
(6)	ABCDEFGHIJ	KLMNOPQRST	UVWXYZ	ABCDEFGHIJ
	KLMNOPQRST	UVWXYZ		

なお、文字列を書く場合は、あとにステートメントが続かないときに限って、後ろの引用符（"）を省略することができます。

（例５）１００　ＰＲＩＮＴ"データ

ＰＲＩＮＴステートメントの最後にセミコロン（;）をつけると、次のＰＲＩＮＴステートメントの表示がすぐ後ろに続きます。

ＰＲＩＮＴを単独に用いると、１行改行が行なわれます。

ＰＲＩＮＴはよく使われるステートメントなので、疑問符？を省略形として使うことができるようになっています。

〈ＴＡＢとＳＰＣ〉

ＰＲＩＮＴステートメント中で、表示位置をスキップするためにＴＡＢ、指定位置まで空白を表示するためにＳＰＣがあります。

（例６）ＰＲＩＮＴ　ＴＡＢ（１０）；"Ａ"

⇨左端から１０文字分あけて文字Ａを表示します。

（例７）ＰＲＩＮＴ"Ａ"；ＳＰＣ（１０）；"Ｂ"

⇨文字Ａを表示後、１０文字分の空白をあけて文字Ｂを表示します。

（例８）

```
10 ' TAB/SPC
20 PRINT 10;TAB(10);"Personal Computer"
30 PRINT"Personal";TAB(10);"Computer"
40 PRINT
50 PRINT 10;SPC(10);"Personal Computer"
60 PRINT"Personal";SPC(10);"Computer"
RUN
 10       Personal Computer
Personal  Computer

 10           Personal Computer
Personal          Computer
Ok
```

２０：数字１０を表示し、画面左端から１０文字分スキップして文字列「Ｐｅｒｓｏｎａｌ　Ｃｏｍｐｕｔｅｒ」を表示します。

３０：文字列「Ｐｅｒｓｏｎａｌ」を表示し、画面左端から１０文字分スキップして文字列「Ｃｏｍｐｕｔｅｒ」を表示します。

４０：改行。

５０：数字の１０、１０文字分の空白、文字列「Ｐｅｒｓｏｎａｌ　Ｃｏｍｐｕｔｅｒ」を表示します。

６０：文字列「Ｐｅｒｓｏｎａｌ」、１０文字分の空白、文字列「Ｃｏｍｐｕｔｅｒ」を表示します。

＊　このように、ＴＡＢ（１０）は画面左端から１０文字スキップするのに対し、ＳＰＣ（１０）はそのまま１０文字分の空白を表示します。

サンプル
プログラム

2.3

(例１)

```
10 ' PRINT (例1)
20 A=45
30 C$="m/sec"
40 PRINT A;C$
50 PRINT A,C$
60 PRINT TAB(5);A
70 PRINT SPC(5);A
80 PRINT A;TAB(5);C$
90 PRINT A;SPC(5);C$
100 PRINT A,A,A
110 PRINT A;A;A
RUN
 45 m/sec
 45            m/sec
      45
      45
 45  m/sec
 45       m/sec
 45            45            45
 45  45  45
Ok
```

(例２)

```
10 ' PRINT (例2)
20 WIDTH 40,25
30 A$="ABC"
40 B$="ABCDEFGHI"
50 C$="ABCDEFGHIJ"
60 D$="ABCDEFGHIJKLM"
70 E$="ABCDEFGHIJKLMNOPQRST"
80 F$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
90 PRINT A$,A$
100 PRINT
110 PRINT B$,B$
120 PRINT
130 PRINT C$,C$
140 PRINT
150 PRINT D$,D$
160 PRINT
170 PRINT E$,E$
180 PRINT
190 PRINT F$,F$
```

2.3.7 PRINT#0

機　能

〔1〕拡大文字を画面に表示します。
〔2〕制御文字を画面に表示します。

書　式

```
PRINT#0〔, x1, x2, ……〕
```

xi：数式または文字式。i＝1，2，3，……。

省略形

P．#0 または ？#0

文　例

CSIZE　3：PRINT#0"SHARP"
⇨「SHARP」のたて横2倍文字が画面に表示されます。

解　説

PRINT#0には、2つの働きがあります。

〔1〕CSIZEステートメントで指定したサイズの拡大文字を画面に表示します。
CSIZE1が指定してあれば、たて2倍文字、
CSIZE2が指定してあれば、横2倍文字、
CSIZE3が指定してあれば、たて横2倍文字
CSIZE0（または単なるCSIZE）が指定してあれば、標準の文字をそれぞれ表示します。

〔2〕PRINTステートメントでは表示できない制御文字を画面に表示します。このとき、制御文字をコントロールコードとして使用することはできません。

(例) PRINT　CHR$（&H0C）⏎…コントロール・コードの&H0Cが働いて画面を消去します。
PRINT#0，CHR$（&H0C）⏎…画面に制御文字が表示されます。

参　照

2.3.6 PRINT、2.7.7 CSIZE

サンプル
プログラム

2.3

(例1) 拡大した文字を表示します。

```
10 ' PRINT#0 (例1)
20 WIDTH40,25:CLS
30 A$="PERSONAL COMPUTER"
40 PRINT A$
50 CSIZE0:PRINT#0 A$
60 CSIZE1:PRINT#0 A$
70 CSIZE2:PRINT#0 A$
80 CSIZE3:PRINT#0 A$
90 CSIZE0
```

(例2) (例2—1) キャラクタを表示します。

```
10 ' PRINT#0 (例2-1)
20 KMODE0
30 FOR I=0 TO 255
40 PRINT CHR$(I);
50 NEXT
60 KMODE1
```

(例2—2) 制御文字までキャラクタを表示します。

```
10 ' PRINT#0 (例2-2)
20 KMODE0
30 FOR I=0 TO 255
40 PRINT#0 CHR$(I);
50 NEXT
60 KMODE1
```

2.3.8 PRINT USING

機能　画面に表示するデータの書式を指定します。

書式

```
PRINT USING"書式指定",式〔,式,……〕
```

書式指定：書式指定子のならび。
式：画面に表示するデータ。数値、文字式。単独の定数、変数、配列も含む。
　カンマ（,）はセミコロン（;）で置き換えることができる。

省略形　P. US. または ? US.

文例　PRINT USING"###.####";X
⇨Xの値を整数部3けた、小数部4けたに揃えて表示します。

解説

PRINT USINGは、画面にデータを表示するときに、数値の小数点の位置を揃えたい場合や、3けたごとにカンマ（,）で区切りたい場合などに使用するステートメントです。

PRINT USINGの後ろに続く引用符（"）で囲まれた文字列に、以下に述べる書式指定子と呼ばれる記号を書くと、それは文字として画面に表示されずに、表示するデータの書式を指定するのに使われます。

書式指定子は大きく分けて、数値型と文字型の2種類あり、次のように細分されます。

(1)数値型

a）#（シャープ）　表示する数字のけた数を指定します。数値のけた数が、指定したけた数より小さいときは、右詰めで表示され、指定したけた数より大きいときは、「Format over」のエラーが出ます。

```
10 ' PRINT USING (例1 #)
20 A=12
30 B=345
40 PRINT USING"######";A;B;6789
RUN
    12   345  6789
Ok
```

b）．（ピリオド）　小数点以下第何位まで表示するか、小数点の位置をどこにするか、を指定します。数値の小数部のけた数が指定より大きいときは、はみ出した部分を四捨五入して表示します。数値の整数部のけた数が指定より大きいときは、「Ｆｏｒｍａｔ　ｏｖｅｒ」のエラーが出ます。

```
10 ' PRINT USING (例2 .)
20 A=.23
30 B=45.678
40 PRINT USING"###.##  ";A;B,12.3,.236
RUN
  0.23   45.68   12.30    0.24
Ok
```

c），（カンマ）　数値に３けたごとにカンマ（,）をいれます。

```
10 ' PRINT USING (例3 ,)
20 A=1000000!
30 B=20000
40 PRINT USING"###,###,###";A;B,100
RUN
  1,000,000     20,000         100
Ok
```

d）＋と－（正負の符号）　＋と－は、数値を表示するときに、＋と－の符号をどこに表示するのか指定します。＋を指定すると、正の数の頭に＋をつけます。＋は＃の列の前と後に、－は＃の列の後のみに指定できます。

```
10 ' PRINT USING (例4 +-)
20 A=235
30 B=-440
40 PRINT USING"#####+";A;B
50 PRINT USING"#####-";A;B
60 PRINT USING"+#####";A;B
RUN
  235+  440-
  235   440-
  +235  -440
Ok
```

e）＊＊　書式指定の先頭に書き、数値の左側の空白の部分を＊で埋めます。＊＊自身もけた数に含まれます。

```
10 ' PRINT USING (例5 **)
20 A=2.35
30 B=-440
40 PRINT USING"**##.#  ";A;B,123.48;.456
7
RUN
***2.4  -440.0  *123.5  ***0.5
Ok
```

f）￥￥　数字の前に円マーク（￥）をつけます。￥自身もけた数に含まれます。

```
10 ' PRINT USING (例6 ¥¥)
20 A=550
30 PRINT USING"¥¥##,###";A;6100,35500!
RUN
    ¥550  ¥6,100 ¥35,500
Ok
```

2.3

g）＊＊￥　この指定は、＊＊と￥￥を組み合わせたもので、数字の前に￥をつけ、数値の左端から￥までの空白を＊で埋めます。

```
10 ' PRINT USING (例7 **¥)
20 A=550
30 PRINT USING"**¥##,###  ";A;6100,35500
!
RUN
*****¥550  ***¥6,100  **¥35,500
Ok
```

h）＾＾＾＾　これをけた数指定の＃の後ろに置くと、数値を浮動小数点表示することができます。単精度型の表現（E表現）、倍精度型の表現（D表現）のいずれも表示できます。

```
10 ' PRINT USING (例8 ^^^^)
20 A=235.3
30 B#=.004403
40 PRINT USING"##.####^^^^";A;B#;10000
RUN
 2.3530E+02 4.4030D-03 1.0000E+04
Ok
```

i）1つのＰＲＩＮＴ　ＵＳＩＮＧ文に複数のデータを並べると、そのデータすべてに対し、書式指定したとみなします。

(2)文字型

a）！（感嘆符）　文字列の先頭の1文字のみ表示します。

```
10 ' PRINT USING (例9 !)
20 A$="YUUKO"
30 B$="TANAKA"
40 PRINT USING"!";A$;B$
RUN
YT
Ok
```

b）＆（空白）＆　文字列を＆から＆までのけた数分表示します。文字列の長さが、指定けた数より小さいときは、余った右の部分が空白で埋まります。

```
10 ' PRINT USING (例10 &  &)
20 A$="YUUKO"
30 B$="TANAKA"
40 PRINT USING"& &";A$;B$
50 PRINT USING"&        &";A$;B$
RUN
YUUTAN
YUUKO   TANAKA
Ok
```

(3)その他

a）＿（アンダーライン）この後ろに伴った書式指定子１個を、単なる１文字とみなして表示します。

```
10 ' PRINT USING (例11 _)
20 A$="YUUKO"
30 B=3
40 PRINT USING"_#&&";A$
50 PRINT USING"_####& &";B;A$
RUN
#YU
#   3YUU
Ok
```

(4)文字列の表字

a）ＰＲＩＮＴ　ＵＳＩＮＧの書式指定子の他の文字がある場合は、それをそのまま表示します。

```
10 ' PRINT USING (例12)
20 A=2350
30 B=9.95
40 PRINT USING"FINAL ¥###,###";A
50 PRINT USING"##.## SEC_.";B
RUN
FINAL ¥  2,350
 9.95 SEC.
Ok
```

2.3.9 LPRINT

機　能

プリンタにデータを印字します。

書　式

```
LPRINT 式〔, 式, ……〕
```

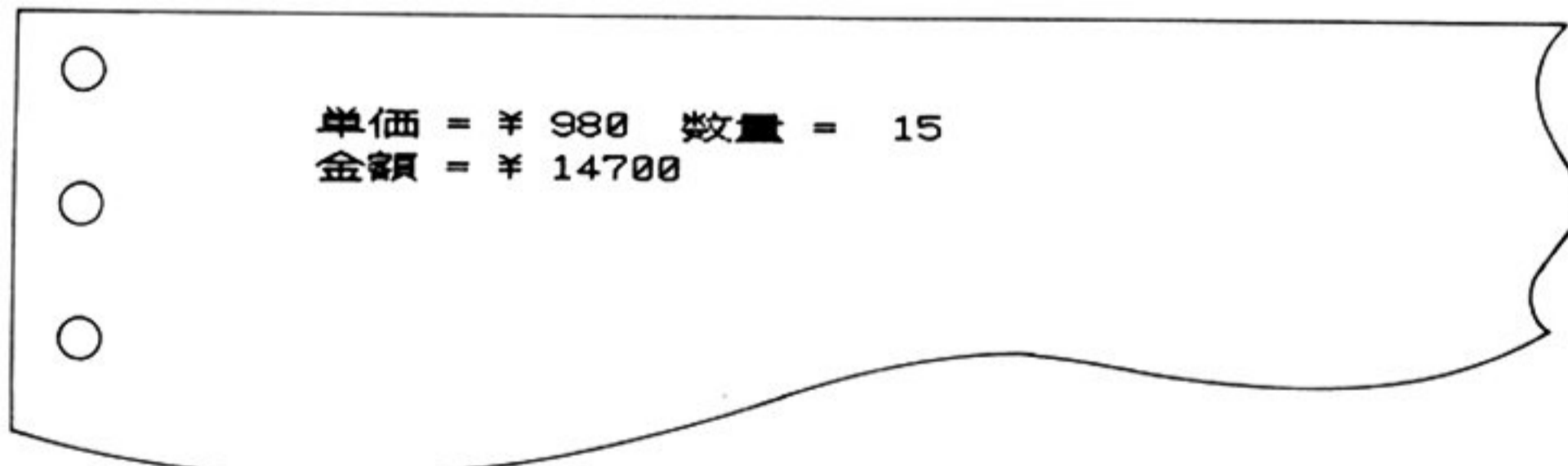
式：数値、文字式。単独の定数、変数、配列を含みます。

省略形

LP.

文　例

LPRINT　A，B，C

⇨プリンタにA，B，Cの値の順で打ち出します。

解　説

LPRINTは、プリンタに定数の値、変数の値および式の値を打ち出します。

PRINTステートメントと同様に、区切り記号としてカンマ（,）やセミコロン（;）を使うことができます。

LPRINTを単独に使うと、1行改行が行なわれます。

参　照

2.3.6 PRINT

サンプルプログラム

```
10 ' LPRINT (例)
20 INPUT"単価 =";A
30 INPUT"数量 =";B
40 C=A*B
50 LPRINT" 単価 = ¥";A;
60 LPRINT" 数量 = ";B
70 LPRINT" 金額 = ¥";C
RUN
単価 =? 980
数量 =? 15
Ok
```

```
単価 = ¥ 980  数量 =  15
金額 = ¥ 14700
```

2.3.10 LPRINT USING

2.3

機　能　プリンタに印字するデータの書式を指定します。

書　式

```
LPRINT USING"書式指定";式〔;式; ……〕
```

書式指定：書式指定子の文字列
式：表示するデータ。数式、または文字式。単独の定数、変数、配列も含みます。
　セミコロン（;）の部分にカンマ（,）も使用できます。

省略形　LP. US.

文　例　LPRINT USING"###.####";X
⇨Xの値を整数部3けた、小数部4けたそろえてプリンタに印字します。

解　説　「PRINT USING」のプリンタ版です。「PRINT USING」を参照してください。

参　照　2.3.8 PRINT USING

2.3.11 WRITE

機能 画面にデータを表示します。

書式

```
WRITE 式〔, 式, ……〕
```

式：数値、文字式。単独の定数、変数、配列を含みます。

省略形 WR.

文例 WRITE A, B, C

⇨画面にA, B, Cの値を順にカンマ（,）で区切って表示します。

解説 WRITEは、画面上に定数の値、数値変数の値、文字変数の値（文字列）、および式の値を表示します。

複数個のデータは、カンマ（,）で区切り、空白なく詰めて表示され、文字列は、引用符（"）で囲んで表示されます。

WRITEステートメントの中に、TAB関数を使うことはできません。

参照 2.3.6 PRINT、2.4.6 WRITE#

サンプルプログラム

（例1）
```
10 ' WRITE (例1)
20 PRINT 0,1,2,3,4,5
30 PRINT 0;1;2;3;4;5
40 WRITE 0,1,2,3,4,5
50 WRITE 0;1;2;3;4;5
RUN
 0           1           2           3
 4           5
 0  1  2  3  4  5
0,1,2,3,4,5
0,1,2,3,4,5
Ok
```

（例2）
```
10 ' WRITE (例2)
20 PRINT "ABC","DEFGH","IJKLMNOPQRSTUVWX
Y","Z"
30 PRINT "ABC";"DEFGH";"IJKLMNOPQRSTUVWX
Y";"Z"
40 WRITE "ABC","DEFGH","IJKLMNOPQRSTUVWX
Y","Z"
50 WRITE "ABC";"DEFGH";"IJKLMNOPQRSTUVWX
Y";"Z"
RUN
ABC          DEFGH        IJKLMNOPQRSTUVWXY
Z
ABCDEFGHIJKLMNOPQRSTUVWXYZ
"ABC","DEFGH","IJKLMNOPQRSTUVWXY","Z"
"ABC","DEFGH","IJKLMNOPQRSTUVWXY","Z"
Ok
```

（例1），（例2） PRINTステートメントとWRITEステートメントの違いを示すプログラムです。

2.3.12 POKE

機　　能　メインメモリに1バイトのデータを書き込みます。

書　　式

```
POKE a, 式〔, 式, …〕
```

ａ：データを書き込むアドレス。０～＆ＨＦＦＦＦ。
式：数式。単独の数値定数。数値変数。ただし０～２５５の整数。

省 略 形　ＰＯ.

文　　例　ＰＯＫＥ　＆ＨＣ０００，１００
⇨メモリ内のアドレス＆ＨＣ０００に数値データ１００を書き込みます。

解　　説　メインメモリ上のアドレスaに直接1バイトのデータを書き込みます。
　データは、０～２５５（＆Ｈ０～＆ＨＦＦ）の値をもつ式で表わされます。式をカンマで区切って続けて書くと、a以降の連続したアドレスに書き込むことができます。
　ＰＯＫＥは、現在のメモリの内容を書き換えてしまうので、ＢＡＳＩＣのシステム領域にＰＯＫＥを行なわないよう十分に気をつけてください。
　Ｉ／Ｏメモリに書き込む場合は、ＯＵＴを使用します。

ＢＡＳＩＣ起動時のメモリマップ

アドレス	領域
＆Ｈ００００	ＢＡＳＩＣのシステム領域
	ユーザーエリア
＆ＨＦ０００	音訓ワークエリア
＆ＨＦ４００	ＢＡＳＩＣワークエリア
＆ＨＦＦＦＦ	

参　　照　3.6.2 ＰＥＥＫ、2.3.13 ＯＵＴ、3.6.3 ＰＥＥＫ@、2.7.10 ＰＯＫＥ@

サンプル
プログラム

メモリのＤ０００番地以降にＰＯＫＥ文でデータを書き込み、ＰＥＥＫ関数で内容を確認してみましょう。

```
10 ' POKE (例)
20 CLEAR &HD000:WIDTH40
30 I=0
40 READ D$
50 D=VAL("&H"+D$)
60 POKE &HD000+I,D
70 IF D=&HC9 THEN 100
80 I=I+1
90 GOTO 40
100 FOR J=&HD000 TO &HD000+I
110 RD=PEEK(J)
120 PRINT HEX$(RD),
130 NEXT
140 END
1000 '
1010 DATA 3E,07      :'LD  A,07H
1020 DATA 01,00,20 :'LD  BC,2000H
1030 DATA ED,79      :'OUT (C),A
1040 DATA 3E,41      :'LD  A,41H
1050 DATA 01,00,30 :'LD  BC,3000H
1060 DATA ED,79      :'OUT (C),A
1070 DATA C9         :'RET
RUN
3E          7           1           0
20          ED          79          3E
41          1           0           30
ED          79          C9
Ok
```

１０００行以降のデータを、ＰＯＫＥステートメントを使用してメモリーの&ＨＤ０００から順番に書き込みこのメモリーデータをＰＥＥＫステートメントを使用して読み出し表示するプログラムです。

2.3

2.3.13 OUT

2.3

機能　1バイトのデータをI／Oポート（メモリも含む）へ送ります。

書式

```
OUT a, 式
```

a：データを送るポートアドレス。0～&HFFFF。
式：数式。単独の数値定数。数値変数。ただし0～255の整数。

文例　OUT &H4000, 25
⇨I／Oポートのアドレス&H4000に数値データ25を書き込みます。

解説　I／Oポートaに直接1バイトのデータを書き込みます。
データは、0～255（&H0～&HFF）の値をもつ式で表わされます。

参照　2.3.12 POKE、3.6.2 PEEK

サンプルプログラム　OUT文でサウンド（波の音）を鳴らします。キー入力があると音が止まります。

```
10 ' OUT <例>
20 OUT &H1C00,6
30 OUT &H1B00,20
40 OUT &H1C00,7
50 OUT &H1B00,&H37
60 OUT &H1C00,8
70 OUT &H1B00,16
80 OUT &H1C00,11
90 OUT &H1B00,0
100 OUT &H1C00,12
110 OUT &H1B00,90
120 OUT &H1C00,13
130 OUT &H1B00,14
140 A$=INKEY$(1)
150 OUT &H1C00,7
160 OUT &H1B00,&H3F
170 OUT &H1C00,8
180 OUT &H1B00,0
```

2.4 ファイル処理ステートメント

2.4.1 INIT

機能 デバイスの初期化を行ないます。

書式

```
INIT〔"デバイス名："〕
```

デバイス名：『ユーザーズマニュアル』の「ファイルについて」参照。

文例 INIT"EMM0："
⇨外部メモリの初期化を行ないます。

解説 INITは、デバイスを初期化するためのコマンドです。
デバイス名の指すデバイスによって、次のような初期化に関する処理が行なわれます。

デバイス名	処　理　の　内　容
0： ｜ 3： F0： ｜ F3：	ディスク中のフォーマット管理テーブルを初期化します。これによって、ディスク中に既存のプログラムおよびファイルはアクセスできなくなります。(ユーティリティプログラム内のイニシャライズでフォーマットを行なったフロッピーディスク以外またはフォーマットの壊れたフロッピーディスクでは初期化しません。「Device I／O error」のエラーが出ます)
HD0： ｜ HD3：	「Bad file descripter」のエラーが出ます。
CAS：	カセットテープを巻き戻し、1本全部の消去を行ないます。
MEM0： MEM1： EMM0： ｜ EMM9：	フォーマットおよび管理テーブルの初期化を行ないます。これによって既存のプログラムおよびファイルはアクセスできなくなります。(この欄のデバイスはINITでのみ初期化できます)
CRT： SCR：	次のプログラムと同じ処理をして、画面の初期化を行ないます。 COLOR 7，0：CGEN：CFLASH：CREV：CSIZE：PALET：CBLACK0：PRW：WINDOW：CONSOLE：SCREEN0，0，0：
KEY： LPT：	「Bad file descripter」のエラーが出ます。

①COLOR　7，0	背景色が黒、表示色は白。
CGEN	ROMCGの文字を表示。
CFLASH	文字を明滅しない。
CREV	文字を反転しない。
CSIZE	標準サイズの文字を表示。
CBLACK　0	カラーコード0の初期化。
PALET	パレットコードの初期化
PRW	グラフィックより文字優先。
WINDOW	グラフィック画面全体の使用。
CONSOLE	テキスト画面全体の使用。
SCREEN　0，0，0	ページ0のアクセス。

ファイルディスクリプタを指定してダイレクトにINITを実行すると、

Are you sure? (y or n)

と、実行の是非をきいてくるので、実行するときは[Y]キーを押し、実行を止めるときは、[N]キーを押してください。
また単にデバイス名を指定しないでINIT⏎を実行すると、実行の是非をきかずに直接

INIT "CRT:"

と同じ動作をし、画面の初期化を行ないます。

参照　『ユーザーズマニュアル』の「ディスプレイモード」

2.4

2.4.2 MAXFILES

機　能　**同時にオープンできるファイルの数を指定します。**

書　式

```
MAXFILES n
```

n：オープンできるファイル番号の最大値。0～15の整数。

省略形　ＭＡ．

文　例　ＭＡＸＦＩＬＥＳ　2
⇨同時に2つのファイルがオープンできます。

解　説　ＭＡＸＦＩＬＥＳは、オープンできるファイル番号の最大値を指定するためのステートメントです。

(例1)　10　ＭＡＸＦＩＬＥＳ　3……ファイル番号1～3の3つのファイルを使用できるようにします。

(例2)　10　ＭＡＸＦＩＬＥＳ　0……ファイルが1つも使用できなくなります。

ＭＡＸＦＩＬＥＳを実行すると、変数がクリアされます。

ＢＡＳＩＣ起動後、このＭＡＸＦＩＬＥＳを実行しなければ、自動的に、

ＭＡＸＦＩＬＥＳ　2

が指定され、ファイル番号が1と2のみ使用することができます。

ＭＡＸＦＩＬＥＳでオープンするファイルの数は必要最小限にしてください。ファイルをオープンするとフリーエリアにある領域を専有しフリーエリアが減少します。

すなわち、ＭＡＸＦＩＥＳによって、ファイルをアクセスするときに使用され、ファイルバッファというエリアがとられます。ファイルバッファの大きさは〔(ＭＡＸＦＩＬＥＳで指定された数)×(256＋16)〕バイトです。

参　照　2.4.3 ＯＰＥＮ、『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」

サンプル
プログラム

```
10 ' MAXFILES (例)
20 INIT:WIDTH40,25
30 MAXFILES2
40 PRINT"ｺﾋﾟｰ元(ﾄﾞﾗｲﾌﾞ0)-->ｺﾋﾟｰ先(
1)"
50 A$=INKEY$(1)
60 INPUT"ﾌｧｲﾙ名";NM$
70 OPEN"R",#1,"0:"+NM$
80 OPEN"R",#2,"1:"+NM$
90 FIELD#1,128 AS A$,128 AS B$
100 FIELD#2,128 AS C$,128 AS D$
110 FOR I=0 TO LOF(1)
120 GET#1,I
130 LSET C$=A$:LSET D$=B$
140 PUT#2,I
150 NEXT
160 CLOSE#1,#2
170 END

RUN


ｺﾋﾟｰ元(ﾄﾞﾗｲﾌﾞ0)-->ｺﾋﾟｰ先(ﾄﾞﾗｲﾌﾞ1)
ﾌｧｲﾙ名? TEST
Ok
```

ファイル名を入力すると，ドライブ0とドライブ1にランダムアクセスファイルをオープンし，ドライブ0の指定されたファイルの内容をドライブ1の同じ名前のファイルに書き込むプログラムです。

2.4

2.4.3 OPEN

機　能　ファイルをオープンし、プログラムで使用できるようにします。

書　式

```
〔1〕OPEN"I", #n, file
〔2〕OPEN"O", #n, file
〔3〕OPEN"A", #n, file
〔4〕OPEN"R", #n, file
〔5〕OPEN"C", #n, "COM:通信パラメータ"
```

n：ファイル番号。1～15の整数。MAXFILESで設定した値まで指定可能です。

file：『ユーザーズマニュアル』の「ファイルについて」参照。ファイル名は省略できます。

I：シーケンシャル入力モードの指定。

O：シーケンシャル出力モードの指定。

A：シーケンシャル出力モードの指定。

R：ランダム入出力モードの指定。

C：シーケンシャルの入出力モードの指定。

通信パラメータ：『ユーザーズマニュアル』の「RS－232Cインターフェイスの使い方」参照。

文　例　OPEN"O", #1, "0:TEST"

⇨3or5インチフロッピーディスク版のドライブ0をファイル番号1の書き込み用シーケンシャルアクセスファイルとしてオープンし、ファイル名TESTで記録できるように設定します。

OPEN"I", #2, "CAS:TEST1"

⇨カセットをファイル番号2の読み込み用シーケルアクセスファイルとしてオープンし、ファイル名TEST1で読み出せるよう設定します。

解　説

OPENを実行すると、ファイルへの入出力ができるようになります。

fileによってファイルディスクリプタとファイル名を指定しますが、ファイル名は省略できます。

ファイル番号は、1～15の整数で、MAXFILESステートメントで設定した値まで指定できます。MAXFILESを実行していない場合は、ファイル番号1と2を指定することができます。

ファイル番号は、ファイルを識別するためにつける番号で、1つのファイル番号に対して1つのファイルが対応しており、この対応はそのファイルがオープンされている間維持されます。この番号は、プログラム中で他のファイル処理ステートメントを使用するときに使うことができます。

ファイルをオープンするときには、必ずI、O、A、R、Cのモードを指定します。

参　照　『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」

サンプルプログラム　データをキーから入力してカセットテープに記録するプログラムです。

```
10 ' OPEN (例)
20 '* データ入力 *
30 MAX=1000
40 DIM A(MAX)
50 INPUT"データ名  =";TITL$
60 N=0
70 INPUT D$
80 IF D$="END" THEN 120
90 A(N)=VAL(D$)
100 N=N+1
110 GOTO 70
120 '* データセーブ *
130 INPUT"ファイル名=";NM$
140 OPEN"O",#1,"CAS:"+NM$
150 PRINT"データ書き込み中"
160 PRINT#1,TITL$
170 PRINT#1,N
180 FOR I=1 TO N
190 PRINT#1,A(I)
200 NEXT
210 PRINT"データ書き込み終了"
220 CLOSE#1
230 END
```

2.4.4 CLOSE

機　能　　**ファイルをクローズして、ファイルの使用を終了します。**

書　式

```
CLOSE〔#n1,〔#n2,……〕〕
```

n_1, n_2, ……：ファイル番号。オープンしたファイルの番号。

省略形　　CLO.

文　例　　CLOSE#1, #2

⇨ファイル番号1と2のファイルをクローズします。

解　説

　CLOSEを実行すると、OPENによってオープンされたファイルをクローズして、ファイルの使用を終了します。

　1度クローズしたファイルは、OPENで再指定しない限り使用できません。

　ファイル番号を指定しないでCLOSEを実行すると、それまでオープンされていたすべてのファイルがクローズされます。

　なお、NEW、RUN、LOAD、ENDを実行したときも、オープンされていたすべてのファイルがクローズされます。

参　照　　2.4.3 OPEN、『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」

サンプルプログラム　　2.4.3 OPENのサンプルプログラム参照。

2.4.5 PRINT#

機　　能　**シーケンシャルアクセスファイルにデータを書き込みます。**

書　　式

```
PRINT#n,〔式, 式, ……〕
```

n：ファイル番号。OPENで指定した番号。1～15の整数。
式：数値、文字列を表わす式。

2.4

省 略 形　P. #　または　?#

文　　例　PRINT#1, A, B, C

⇨ファイル番号1の書き込み用シーケンシャルアクセスファイルに数値変数A、B、Cの値を書き込みます。

解　　説　PRINT#は、ファイル番号で指定したファイルに、定数の値、数値変数の値、文字変数の値（文字列）、および式の値を書き込みます。このとき、データは、PRINTステートメントによって画面上に表示するときと同じイメージで、ファイルに書き込まれます。

式はカンマ（,）で区切って、複数個続けて書くことができます。

このステートメントを実行するためには、前もってOPENでファイルをオープンしておく必要があります。また、PRINT#による書き込みが終わったら、必ずCLOSEを実行して、ファイルをクローズしなければなりません。

〈注　　意〉　PRINT#0は、画面上に拡大文字（たて2倍文字、横2倍文字、たて横2倍文字）を表示するためのステートメントです。まちがえないように注意してください。

参　　照　『ユーザーズマニュアル』の「プログラムデータファイルの保存と再生」、2.4.6 WRITE#、2.3.6 PRINT、2.4.7 INPUT#、2.3.7 PRINT#0

サンプル
プログラム

データ名を入力し、次にデータを入力します。データの入力の終了時に"ＥＮＤ"と入力します。次にファイル名を入力し、そのファイル名でカセットに、データ名とデータを書き込み、終了します。

```
10 ' PRINT# (例)
20 MAX=1000:N=0
30 DIM A(MAX)
40 INPUT"データ名  =";TITL$
50 INPUT D$
60 IF D$="END" THEN "データセーブ"
70 N=N+1
80 A(N)=VAL(D$)
90 GOTO 50
100 LABEL"データセーブ"
110 INPUT"ファイル名=";NM$
120 OPEN"O",#1,"CAS:"+NM$
130 PRINT"PRINT#文でデータ書き込み中"
140 PRINT#1,TITL$
150 PRINT#1,N
160 FOR I=1 TO N
170 PRINT#1,A(I)
180 NEXT
190 PRINT"データ書き込み終了"
200 CLOSE#1
210 END
```

2.4.6 WRITE#

機能 シーケンシャルアクセスファイルにデータを書き込みます。

書式

```
WRITE#n,〔式, 式, ……〕
```

n：ファイル番号。OPENで指定した番号。1～15の整数。
式：数値、文字列を表わす式。

省略形 WR. #

文例 WRITE#1, A, B, C
⇨ファイル番号1の書き込み用シーケンシャルアクセスファイルに数値データA, B, Cの値をカンマ(,)で区切って書き込みます。

解説 WRITE#は、ファイル番号で指定したファイルに、定数の値、数値変数の値、文字変数の値(文字列)、および式の値を書き込みます。

WRITE#は、シーケンシャルアクセスファイルにデータを書き込む点ではPRINT#と同じです。

しかし、PRINT#でデータを書き込むと、余分な空白が入ることが多く、データ間の区切りがはっきりしていないので、文字列の場合、INPUT#で読み込めるようにするためには、区切りにカンマ(,)をつけたり、各文字列を引用符(")で囲む必要があります。

これに対してWRITE#は、データを書き込む際に、自動的に余分な空白をすべて省き各データ間に区切り記号のカンマ(,)を入れ、文字列のときは各文字列を引用符(")で囲むのでファイルの使用領域を節約することができます。

このステートメントを実行するためには、前もってOPENでファイルをオープンしておく必要があります。また、WRITE#による書き込みが終わったら、必ずCLOSEを実行して、ファイルをクローズしなければなりません。

参照 『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」、2.3.11 WRITE、2.4.5 PRINT#、2.4.7 INPUT#

サンプル
プログラム

```
10 ' WRITE# (例)
20 MAX=1000:N=0
30 DIM A(MAX)
40 INPUT"データ名  =";TITL$
50 INPUT D$
60 IF D$="END" THEN "データセーブ"
70 A(N)=VAL(D$)
80 N=N+1
90 GOTO 50
100 LABEL"データセーブ"
110 INPUT"ファイル名=";NM$
120 OPEN"O",#1,"CAS:"+NM$
130 PRINT"WRITE#文でデータ書き込み中"
140 WRITE#1,TITL$
150 WRITE#1,N
160 FOR I=1 TO N
170 WRITE#1,A(I)
180 NEXT
190 PRINT"データ書き込み終了"
200 CLOSE#1
210 END
```

データ名を入力し，次にデータを入力します。("ＥＮＤ"を入力すると終了します)
次にファイル名を入力し，そのファイル名でカセットに、データ名とデータを書き込みます。

2.4.7 INPUT＃

機　能　シーケンシャルアクセスファイルからデータを読み込みます。

書　式

```
INPUT#n, 変数〔, 変数, ……〕
```

n：ファイル番号。OPENで指定した番号。1～15の整数。
変数：読み込んだデータの入る整数。

2.4

省略形　I. ＃

文　例　INPUT＃1, A, B, C
⇨ファイル番号1の読み込み用シーケンシャルアクセスファイルから数値データを順次読み込んで変数A、B、Cに入れます。

解　説　ファイル番号で指定したファイルからデータをシーケンシャル（記録されている順）に読み込んで変数に入れます。
変数の型と、ファイル中のデータの型とは一致していなければなりません。
このステートメントを実行するためには、前もってファイルをシーケンシャル入力モードでオープン（OPEN）しておく必要があります。
INPUT＃は、データを読み込むという点では、INPUTとほとんど同じものと考えることができます。違うところは、データの読み込みが、キーボードからだけでなく、シーケンシャル入力モードでオープできるファイルがあれば、どのデバイスからでもできること、画面に「？」が表示されないことの2つです。
ファイルに記録されているデータを読み尽くしたあと、さらにデータを読み込もうとすると、「Input past end」のエラーがでます。

参　照　『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」、2.3.1 INPUT、2.4.8 LINPUT＃、2.4.5 PRINT＃

サンプルプログラム

```
10 ' INPUT# (例)
100 MAX=100
110 DIM A(MAX)
120 INPUT"ファイル名=";NM$
130 OPEN"I",#1,"CAS:"+NM$
140 PRINT"データ読み出し中"
150 INPUT#1,TITL$
160 INPUT#1,N
170 FOR I=1 TO N
180 INPUT#1,A(I)
190 PRINT A(I);
200 NEXT
210 PRINT:PRINT"データ読み出し終了"
220 CLOSE#1
230 END
```

ファイル名を入力するとカセットのそのファイル名の内容を読み出し，表示します。

2.4.8 ＬＩＮＰＵＴ＃／ＬＩＮＥ　ＩＮＰＵＴ＃

機能

シーケンシャルアクセスファイルから１行分（半角文字で２５５文字まで）のデータを読み込みます。

書式

```
〔1〕ＬＩＮＰＵＴ＃n, x＄
〔2〕ＬＩＮＥ　ＩＮＰＵＴ＃n, x＄
```

n：ファイル番号。ＯＰＥＮで指定した番号。１～１５の整数。
x＄：読み込んだ文字列データが入る変数。文字変数。

省略形

〔1〕ＬＩＮ．＃　〔2〕ＬＩＮＥＩ．＃

文例

ＬＩＮＰＵＴ＃１，Ａ＄
⇨ファイル番号１の読み込み用シーケンシャルアクセスから１行分の文字列データを読み込んで変数Ａ＄に入れます。

解説

ファイル番号で指定したファイルから１行分（半角文字で２５５文字以内）の文字列データをシーケンシャル（記録されている順）に読み込んで文字変数に入れます。

このとき、文字列データにカンマ（,）、コロン（：）、セミコロン（;）、引用符（"）が含まれていても、それは文字列の一部とみなされます。１行分の文字列の終わりは、キャリッジリターンコード（&Ｈ０Ｄ）とします。このステートメントを実行するためには、前もってシーケンシャル入力モードでファイルをオープン（ＯＰＥＮ）しておく必要があります。

ＬＩＮＰＵＴ＃（ＬＩＮＥ　ＩＮＰＵＴ＃）は、文字列データを１行読み込むという点では、ＬＩＮＰＵＴ（ＬＩＮＥ　ＩＮＰＵＴ）とほとんど同じものと考えることができます。違うのは、データの読み込みが、キーボードからだけでなく、シーケンシャル入力モードでオープンできるファイルであれば、どのデバイスからでもできる点です。

参照

『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」、2.3.2 ＬＩＮＰＵＴ、2.4.7 ＩＮＰＵＴ＃

サンプル
プログラム

```
10 ' LINPUT# (例)
20 '* データ入力 *
30 MAX=100
40 DIM A$(MAX)
50 N=0
60 LINPUT D$
70 IF D$="END" THEN 110
80 A$(N)=D$
90 N=N+1
100 GOTO 60
110 INPUT"ファイル名=";NM$
120 OPEN"O",#1,"0:"+NM$
130 PRINT"データ書き込み中"
140 PRINT#1,N
150 FOR I=0 TO N
160 PRINT#1,A$(I)
170 NEXT
180 PRINT"データ書き込み終了"
190 CLOSE#1
200 K$=INKEY$(1)
210 REW
220 INPUT"ファイル名=";NM$
230 OPEN"I",#1,"0:"+NM$
240 PRINT"データ読み出し中"
250 INPUT#1,N
260 FOR I=0 TO N
270 LINPUT#1,A$(I)
280 PRINT A$(I)
290 NEXT
300 PRINT"データ読み出し終了"
310 CLOSE#1
320 END
```

データを入力し（"ＥＮＤ"と入力すると終了）ファイル名を入力してドライブ0にそのファイル名で，先に入力したデータを書き込みます。
次に何かキー入力しますと，ファイル名を聞いてきますので入力します。
すると，ドライブ0の入力したファイル名のファイルからデータを読み出し表示します。入力するファイル名を同じくすると先に入力したデータが表示されます。

2.4.9 FIELD

機　能　ファイルバッファ内に変数を割りつけます。

書　式

```
FIELD#n, f1 AS x$〔, f2 AS y$, ……〕
```

n：ファイル番号。OPENで指定した番号。1～15の整数。
f_1, f_2, ……：フィールド幅。1～255の整数。バイト数で指定。
x$, y$, ……：ファイルバッファに割りつけられる文字変数。

2.4

省略形　FI.

文　例　FIELD#1, 20 AS N$, 30 AS T$

⇨ファイル番号1のファイルバッファ256バイトのうち20バイトをN$に、30バイトをT$に割りつけて、残りの206バイトは使用しないと定義します。

解　説

FIELDは、PUTやGETステートメントでランダムアクセスファイルをアクセスするときに使用される変数を定義するためのステートメントです。

FIELDを実行すると、1レコード（256バイト）あるファイルバッファをどのように区切って文字変数に割りつけるかが設定されます。

ランダムアクセスファイルのアクセス（書き込み／読み出し）は、1レコード（＝256バイト）という単位で行なわれます。すなわち、PUTによってデータをランダムアクセスファイルに書き込むと、ファイルバッファからランダムアクセスファイルに1レコード分のデータが送られ、GETによってデータをランダムアクセスファイルから読み出すと、ランダムアクセスファイルからファイルバッファに1レコード分のデータが送られてきます。

(例) FIELD#1, 16 AS NM$, 24 AS TL$

……ファイル番号1のファイルバッファ256バイトのうち、16バイトをNM$に24バイトをTL$に割りつけて、残りの216バイトは使用しない、と定義しています。

1レコード中、文字変数の割り当てられる領域をフィールドといい、1つの文字変数に割りつけられたバイト数をフィールド幅といいます。フィールド幅の最大値は255です。

(例) FIELD#1, 16 AS NM$, 24 AS TL$

…… NM$のフィールド幅は16、TL$のフィールド幅は24です。

FIELDは、同じファイル番号のファイルバッファに対して何度でも実行することができます。

FIELDによって定義した文字変数の変数名を、プログラム中で代入文の左辺に使用したり、INPUTステートメントなどの入出力ステートメントに使用すると、それ以降その変数名は普通の文字変数とみなされ、ランダムアクセスファイルのアクセスには使用できなくなるので注意してください。

このステートメントを実行するためには、前もってファイルをランダム入出力モードでオープンしておく必要があります。

参照

『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」、2.4.12 PUT、2.4.13 GET、2.4.3 OPEN、2.4.2 MAXFILES

サンプルプログラム

（例1－1）の60行と（例1－2）の60～64行は同じ意味になります。

（例1－1）

```
10 ' FIELD (例1-1)
20 INIT:WIDTH40
30 MAXFILES1
40 INPUT"ファイル名=";NM$
50 OPEN"R",#1,"1:"+NM$
60 FIELD#1,24 AS N$,24 AS T$,24 AS B$
70 LABEL"入力":N=N+1
80 PRINT
90 INPUT"氏名=";X$
100 IF X$="END" OR X$="end" OR X$="ｵﾜﾘ"
THEN "クローズ"
110 INPUT"電話=";Y$
120 INPUT"住所=";Z$
130 LSET N$=X$
140 LSET T$=Y$
150 LSET B$=Z$
160 PUT#1,N
170 GOTO"入力"
180 LABEL"クローズ"
190 CLOSE#1
200 END
```

（例1－2）

```
10 ' FIELD (例1-2)
20 INIT:WIDTH40
30 MAXFILES1
40 INPUT"ファイル名=";NM$
50 OPEN"R",#1,"1:"+NM$
60 FIELD#1,24 AS N$
62 FIELD#1,24 AS DUMMY$,24 AS T$
64 FIELD#1,48 AS DUMMY$,24 AS B$
70 LABEL"入力":N=N+1
80 PRINT
90 INPUT"氏名=";X$
100 IF X$="END" OR X$="end" OR X$="ｵﾜﾘ"
THEN "クローズ"
110 INPUT"電話=";Y$
120 INPUT"住所=";Z$
130 LSET N$=X$
140 LSET T$=Y$
150 LSET B$=Z$
160 PUT#1,N
170 GOTO"入力"
180 LABEL"クローズ"
190 CLOSE#1
200 END
```

2.4.10 ＬＳＥＴ

機　　能　　**ファイルバッファにデータを書き込みます。**

書　　式

```
ＬＳＥＴ　ｘ＄＝文字式
```

ｘ＄：文字変数。ＦＩＥＬＤで定義した文字変数。
文字式：文字列を表す式。単独の文字列、文字変数を含む。

2.4

省 略 形　　ＬＳ．

文　　例　　ＬＳＥＴ　Ａ＄＝"ＰＲＥＴＴＹ"

⇨ファイルバッファに割りつけられた文字変数Ａ＄に文字列ＰＲＥＴＴＹを書き込みます。

解　　説

ＬＳＥＴは、ランダムアクセスファイルにＰＵＴで文字列データを書き込む１つ前の段階として、そのデータをファイルバッファに置くためのステートメントです。

ＬＳＥＴの実行に先立って、ＦＩＥＬＤでファイルバッファに文字変数ｘ＄を割りつけておく必要があります。

ＬＳＥＴ　ｘ＄＝文字式

において、等号の右側は書き込みたい文字列データを表わす文字式で、左側のｘ＄はＦＩＥＬＤですでに定義してある文字変数です。文字式の表わす文字列の長さがｘ＄のフィールド幅より小さいとき、その文字列はフィールドに左詰めで置かれ、フィールドの余った領域には空白が入れられます。逆に、文字列の長さがフィールド幅より大きいときは、右側のはみ出た部分が失われます。

数値をファイルバッファに書き込むには、ＬＳＥＴする前にＭＫＩ＄、ＭＫＳ＄、ＭＫＤ＄関数を使って文字型に変換しなければなりません。

ＬＳＥＴは、ＦＩＥＬＤで定義していない普通の文字変数に対しても使うことができます。

このステートメントを実行するためには、前もってファイルをランダム入出力モードでオープンしておく必要があります。

参　　照　　『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」、
2.4.11 ＲＳＥＴ、2.4.12 ＰＵＴ、2.4.9 ＦＩＥＬＤ、2.4.3 ＯＰＥＮ、
3.2.17 ＭＫＩ＄、3.2.17 ＭＫＳ＄、3.2.17 ＭＫＤ＄

サンプルプログラム

ファイル名を入力すると、"氏名="のとき"END"、または"end"、または"オワリ"が入力されるまで終了せず、"氏名="、"電話="、"住所="と聞いてくるのでデータを入力してください。

このファイルは、ドライブ1に書き込まれます。

```
10 ' LSET (例)
20 INIT:WIDTH40
30 MAXFILES1
40 INPUT"ファイル名=";NM$
50 OPEN"R",#1,"1:"+NM$
60 FIELD#1,24 AS N$,24 AS T$,24 AS B$
70 LABEL"入力":N=N+1
80 PRINT
90 INPUT"氏名=";X$
100 IF X$="END" OR X$="end" OR X$="オワリ"
THEN "クローズ"
110 INPUT"電話=";Y$
120 INPUT"住所=";Z$
130 LSET N$=X$
140 LSET T$=Y$
150 LSET B$=Z$
160 PUT#1,N
170 GOTO"入力"
180 LABEL"クローズ"
190 CLOSE#1
200 END
```

2.4

2.4.11　RSET

機　　能

〔1〕ファイルバッファにデータを書き込みます。
〔2〕文字変数に右詰めで文字列を代入します。

書　　式

```
RSET　x$=文字式
```

x$：文字変数。FIELDで定義した文字変数。
文字式：文字列を表す式。単独の文字列、文字変数を含む。

省 略 形

RS.

文　　例

RSET　A$="PRETTY"
⇨ファイルバッファに割りつけられた文字変数A$に文字列"PRETTY"を書き込みます。

解　　説

〔1〕RSETは、ランダムアクセスファイルにPUTで文字列データを書き込む1つ前の段階として、そのデータをファイルバッファに置くためのステートメントです。RSETの実行に先立って、FIELDでファイルバッファに文字変数x$を割りつけておく必要があります。

RSET　x$=文字式

において、等号の右側は書き込みたい文字列データを表わす文字式で、左側のx$はFIELDですでに定義してある文字変数です。文字式の表わす文字列の長さがx$のフィールド幅より小さいとき、その文字列はフィールドに右詰めで置かれ、フィールドの余った領域には空白が入れられます。逆に、文字列の長さがフィールド幅より大きいときは、右側のはみ出た部分が失われます。

数値をファイルバッファに書き込むには、RSETする前にMKI$、MKS$、MKD$関数を使って文字型に変換しなければなりません。

このステートメントを実行するためには、前もってファイルをランダム入出力モードでオープンしておく必要があります。

〔2〕RSETは、FIELDで定義していない普通の文字変数に対しても使うことができます。

(例)
```
100　INPUTX$
110　A$=SPACE$(32)
120　RSET　A$=X$
```
………文字変数X$を32文字のフィールド内で右詰めにする。

参　　照

『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」、
2.4.10 LSET、2.4.12 PUT、2.4.9 FIELD、2.4.3 OPEN、
3.2.17 MKI$、3.2.17 MKS$、3.2.17 MKD$

サンプル
プログラム

```
10 ' RSET (例)
20 INIT:WIDTH40
30 MAXFILES1
40 INPUT"ファイル名=";NM$
50 OPEN"R",#1,"1:"+NM$
60 FIELD#1,24 AS N$,24 AS T$,24 AS B$
70 LABEL"入力":N=N+1
80 PRINT
90 INPUT"氏名=";X$
100 IF X$="END" OR X$="end" OR X$="オワリ"
THEN "クローズ"
110 INPUT"電話=";Y$
120 INPUT"住所=";Z$
130 RSET N$=X$
140 RSET T$=Y$
150 RSET B$=Z$
160 PUT#1,N
170 GOTO"入力"
180 LABEL"クローズ"
190 CLOSE#1
200 END
```

ＬＳＥＴステートメントのサンプルと同じ動作です。

2.4.12　PUT

機　能　ランダムアクセスファイルにデータを書き込みます。

書　式

```
PUT〔#〕n〔, r〕
```

n：ファイル番号。OPENで指定した番号。1～15の整数。
r：レコード番号。0～32767の整数を値にもつ数式。

文　例　PUT#1, 2
⇨ファイルバッファからファイル番号1のランダムアクセスファイルの2番レコードに1レコード分のデータを書き込みます。

解　説　ファイルバッファの1レコード（256バイト）分のデータを、ランダムアクセスファイル内の指定したレコードに書き込みます。
これに先立って、FIELDでファイルバッファに文字変数を定義し、LSETかRSETで文字列データを書き込んでおく必要があります。
レコード番号rを省略すると、このステートメントの前までにPUTで書き込んだレコードか、GETで読み出したレコードの次のレコードに書き込まれます。
このステートメントを実行するためには、前もってファイルをランダム入出力モードでオープンしておく必要があります。
なお、カセットテープに対しては使用できません。

参　照　『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」、2.4.10 LSET、2.4.11 RSET、2.4.9 FIELD、2.4.3 OPEN

サンプルプログラム　2.4.10 LSETのサンプルプログラム参照。

2.4.13 GET

機能

ランダムアクセスファイルからデータを読み込みます。

書式

```
GET 〔#〕n 〔, r〕
```

n：ファイル番号。OPENで指定した番号。1～15の整数。
r：レコード番号。0～32767の整数を値にもつ数式。

文例

GET#1, 2
⇨ファイル番号1のランダムアクセスファイルの2番レコードからファイルバッファへ1レコード分のデータを読み込みます。

解説

ランダムアクセスファイルの指定されたレコードからファイルバッファへ1レコード（256バイト）分のデータを読み込みます。

このとき、前もってFIELDでファイルバッファに文字変数を定義しておかなければなりません。

レコード番号rを省略すると、その前にGETで読み込んだレコードか、PUTで書き込んだレコードの次のレコードから読み込まれます。

このステートメントを実行するためには、前もってファイルをランダム入出力モードでオープンしておく必要があります。

なお、カセットテープに対しては使用できません。

参照

『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」、2.4.12 PUT、2.4.9 FIELD、2.4.3 OPEN

サンプルプログラム

入力されたファイル名のファイルから（ドライブ1の）データを読み出し、表示します。PUTステートメントで作ったファイルを使うとよいでしょう。

```
10 ' GET <例>
20 INIT:WIDTH40
30 MAXFILES1
40 INPUT"ファイル名=";NM$
50 OPEN"R",#1,"1:"+NM$
60 FIELD#1,24 AS N$,24 AS T$,24 AS B$
70 LABEL"読み出し":N=N+1
80 IF LOF(1)=N-1 THEN "クローズ"
90 PRINT"GET文でデータを読み出します。"
100 GET#1,N
110 PRINT
120 PRINT"氏名=";N$
130 PRINT"電話=";T$
140 PRINT"住所=";B$
150 GOTO"読み出し"
160 LABEL"クローズ"
170 CLOSE#1
180 END
```

2.4.14 DEVI$

機能　**指定したデバイスから1レコード（256バイト）分のデータを読み込みます。**

書式

```
DEVI$"デバイス名", r, x$, y$
```

r：レコード番号。0～40391の整数を値にもつ数式。
x$，y$：文字列データを入れる変数。

文例　DEVI$"EMM0：", 1, A$, B$
⇨外部メモリ0の1番レコードから文字列データを読み込み、文字変数A$とB$にそれぞれ128バイトずつ入れます。

解説　指定したデバイスから1レコード（256バイト）分の文字列データを読み込みます。

rは、レコード番号を指定する値で、0～40391の整数です。

読み込まれた1レコード分の文字列データは、128バイトずつ2つの文字変数に分けて入れられます。

指定できるデバイス名は、次の通りです。

CAS：……	カセット
0：～3：……	3 or 5インチ版ディスク
F0：～F3：……	8インチ版ディスク
HD0：～HD3：……	ハードディスク
MEM0：～MEM1：…	グラフィックメモリ
EMM0：～EMM9：…	外部メモリ

参照　『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」、2.4.15 DEVO$

2.4

2.4.15　DEVO$

機　能　**指定したデバイスへ１レコード（２５６バイト）分のデータを書き込みます。**

書　式

```
DEVO$ "デバイス名", r, 文字式, 文字式
```

r：レコード番号。0～４0３９１の整数を値にもつ数式。
文字式：文字列を表す式。１２８文字。

2.4

文　例　DEVO$ "EMM0：", 1, A$, B$
⇨外部メモリ0の１番レコードに２つの文字列Ａ＄、Ｂ＄のデータをそれぞれ１２８バイトずつ書き込みます。

解　説　指定したデバイス１レコード（２５６バイト）分の文字列データを書き込みます。このとき、１つの文字式には、１２８バイトの文字が入っている必要があります。
rは、レコード番号を指定する値で0～４0３９１の整数です。
指定できるデバイス名は次の通りです。

ＣＡＳ：……………………	カセット
0：～３：…………………	３ｏｒ５インチ版ディスク
Ｆ0：～Ｆ３：……………	８インチ版ディスク
ＨＤ0：～ＨＤ３：………	ハード・ディスク
ＭＥＭ0：～ＭＥＭ１：…	グラフィック・メモリ
ＥＭＭ0：～ＥＭＭ９：…	外部メモリ

参　照　『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」、2.4.14ＤＥＶＩ＄

2.5 エラー処理ステートメント

2.5.1 ON ERROR GOTO

機能

エラーが発生したとき、指定のエラー処理ルーチンにジャンプします。

書式

```
〔1〕ON  ERROR  GOTO  n
〔2〕ON  ERROR  GOTO  L$
```

n：ジャンプ先の行番号。
L$：ジャンプ先のラベル名。"文字列"。LABELで定義したもの。

2.5

省略形

ON ERR. G.

文例

ON ERROR GOTO 1000
⇨プログラム実行中エラーが発生すると、行番号1000から始まる処理ルーチンにジャンプします。

解説

このステートメントをプログラムの先頭部分で宣言しておくと、プログラム中で、エラーが発生したときに、エラーメッセージを表示せずに、指定の行番号かラベルにジャンプして、それ以降のプログラムを実行します。

一般に、ジャンプ先以降に、エラー対策のプログラムを書くので、エラーの発生したメインプログラムに対して、その部分をエラー処理ルーチンといいます。

エラー処理ルーチンの最後にRESUMEステートメントを書くと、メインプログラムに戻ることができます。

ON ERROR GOTO 0とすると、エラー処理ルーチンへの分岐が禁止となり、通常のエラー処理、すなわち、実行を停止してエラーメッセージを表示します。

また、エラー処理ルーチン中にON ERROR GOTO 0を入れると、それ以降のエラー処理ルーチンが実行されず、通常のエラー処理を行ないます。

エラー処理ルーチン内で発生したエラーは、それに対するエラーメッセージが表示され、実行を停止します。

参照

2.5.2 RESUME、3.10.4 ERL、3.10.5 ERR

サンプルプログラム

(例1)

```
10 ' ON ERROR GOTO (例1)
20 ON ERROR GOTO"ｴﾗｰ"
30 INPUT"N=";N
40 IF N=-1 THEN END
50 S=SUM(N)
60 PRINT"SUM(N)=";S
70 GOTO30
80 LABEL"ｴﾗｰ"
90 IF ERR=5 THEN PRINT"計算できません。"

100 IF ERR=6 THEN PRINT"オーバーフローで
す。"
110 RESUME30
RUN
N=? 10
SUM(N)= 55
N=? 100
SUM(N)= 5050
N=? 1000
SUM(N)= 500500
N=? -1000
計算できません。
N=? 1000000000000000000000
オーバーフローです。
N=? -1
Ok
```

エラー処理ルーチンにジャンプするプログラムです。

(例2)

```
10 ' ON ERROR GOTO (例2)
20 ON ERROR GOTO"ｴﾗｰ"
30 INPUT"N=";N
40 IF N=-1 THEN END
50 S=SUM(N)
60 PRINT"SUM(N)=";S
70 GOTO30
80 LABEL"ｴﾗｰ"
90 ON ERROR GOTO 0
100 IF ERR=5 THEN PRINT"計算できません。
"
110 IF ERR=6 THEN PRINT"オーバーフローで
す。"
120 RESUME30
RUN
N=? 10
SUM(N)= 55
N=? 100
SUM(N)= 5050
N=? 1000
SUM(N)= 500500
N=? -1000
Illegal function call in 50
Ok
```

ON　ERROR　GOTO0を使ったプログラムです。

2.5

2.5.2 RESUME

機能 エラー処理ルーチンからメインプログラムに実行を戻します。

書式

```
〔1〕RESUME
〔2〕RESUME n
〔3〕RESUME L$
〔4〕RESUME NEXT
```

n：エラー処理ルーチン終了後の戻り先を示す行番号。
L＄：エラー処理ルーチン終了後の戻り先を示すラベル名。ＬＡＢＥＬで定義されたもの。
ＮＥＸＴ：エラーが発生した次のステートメントを指します。

2.5

省略形 ＲＥＳＵ.

文例
ＲＥＳＵＭＥ
⇨エラーの発生したステートメントに戻ります。
ＲＥＳＵＭＥ　２００
⇨行番号２００番に戻ります。
ＲＥＳＵＭＥ"Ｅｎｔｒｙ"
⇨ラベル名"Ｅｎｔｒｙ"に戻ります。
ＲＥＳＵＭＥ　ＮＥＸＴ
⇨エラーの発生した次のステートメントに戻ります。

解説
ＯＮ　ＥＲＲＯＲ　ＧＯＴＯによって、エラー処理ルーチンに入った後、メインプログラムに戻るには、エラー処理ルーチンの最後にＲＥＳＵＭＥステートメントを書かなければなりません。

(例1) １９９０　ＲＥＳＵＭＥ……エラーの発生したステートメントに戻ります。ほとんど意味がありません。
(例2) １９９０　ＲＥＳＵＭＥ　２００……２００番に戻ります。
(例3) １９９０　ＲＥＳＵＭＥ　"Ｅｎｔｒｙ"……ＬＡＢＥＬ"Ｅｎｔｒｙ"に戻ります。
(例4) １９９０　ＲＥＳＵＭＥ　ＮＥＸＴ……エラーの発生した次のステートメントに戻ります。

ＲＥＳＵＭＥではなく、ＧＯＴＯステートメントでプログラムに戻ると、再びエラーが発生したとき、エラー処理ができなくなります。

参照 2.5.1 ＯＮ　ＥＲＲＯＲ　ＧＯＴＯ

サンプルプログラム ＯＮ　ＥＲＲＯＲ　ＧＯＴＯ　参照

2.5.3 ERROR

機能 仮のエラーを発生させます。

書式

```
ERROR c
```

C：エラーコード。1～255の整数。

省略形 ERR.

文例 ERROR 2

⇨エラーコード2番の「Syntax error」を発生させます。

解説 エラー処理ルーチンが作ってあっても、エラーが発生しなければ、それを実行しないので正常に動作するかどうかわかりません。そこで、デバックのためにプログラム中にERRORステートメントを入れて、仮のエラーを発生させ、エラー処理ルーチンが正常に作動するかどうか確認します。

ERRORは、指定したエラーコードに対応するエラーを発生させます。

ERRORステートメントは、エラー処理ルーチンのデバッグが終わったら、プログラム中から取り除いてください。

ERRORステートメントをダイレクトに実行した場合は、ERRやERL変数には値が入りません。

1）エラー処理ルーチン2.5.1「ON ERROR GOTO」参照。

参照 「エラーメッセージ表」、2.5.1 ON ERROR GOTO

サンプルプログラム これは25行でエラーを発生させエラー処理ルーチンにジャンプするプログラムです。

```
10 ' ERROR <例>
20 ON ERROR GOTO"ｴﾗｰ"
30 INPUT"N=";N
40 IF N=-1 THEN END
50 S=SUM(N)
60 PRINT"SUM(N)=";S
70 GOTO30
80 LABEL"ｴﾗｰ"
90 IF ERR=5 THEN PRINT"計算できません。"

100 IF ERR=6 THEN PRINT"オーバーフローです。"
110 RESUME30
Ok
25 ERROR 5
RUN
計算できません。
N=? -1
Ok
```

2.6 画面制御ステートメント

2.6.1 SCREEN／GRAPH／SCREEN@／GRAPH@

機能

画面の使用モードを設定します。

書式

```
〔1〕SCREEN  o, i
〔2〕GRAPH  o, i
〔3〕SCREEN  o, i, g
〔4〕GRAPH  o, i, g
〔5〕SCREEN@  o, g, c1, c2
〔6〕GRAPH@  o, g, c1, c2
```

o：出力ページ（表示するページ）。0～3の整数。
i：入力ページ（書き込むページ）。0～3の整数。
g：グラフィックモード。0～3の整数。ただし〔5〕〔6〕については1～3の整数
c_1：表示色の指定、カラーコード。0～7の整数。
c_2：背景色の指定、カラーコード。0～7の整数。
0：黒、1：青、2：赤、3：マゼンタ、4：緑、5：シアン、6：黄、7：白

省略形

〔1〕〔3〕SC.
〔2〕〔4〕GR.
〔5〕SC.@
〔6〕GR.@

文例

SCREEN 0, 1, 1
⇨画面出力ページを0、入力ページを1とし、グラフィック画面1だけドットをセット／リセットできるようにします。
SCREEN@ 0, 1, 7, 0
⇨画面出力ページ0のグラフィック画面1だけ表示します。その時の表示色が白で背景色は黒となります。

解説

SCREEN、SCREEN@、GRAPHおよびGRAPH@はいずれも画面の使用モードを指定するステートメントです。使用できる画面の数は、画面の解像度により、次のようになっています。

2.6

画　面　解　像　度		使用できる画面の数	
横	たて	グラフィック画面	テキスト画面
４０文字 （３２０ドット）	２００ライン 又は１９２ライン	4	2
８０文字 （６４０ドット）	２００ライン 又は１９２ライン	2	1
４０文字 （３２０ドット）	４００ライ 又は３８４ライン	2	2
８０文字 （６４０ドット）	４００ライン 又は３８４ライン	1	1

〔I〕パラメータのｏとｉはそれぞれ出力ページと入力ページを指定します。出力ページとは実際に画面に表示されていて目に見えるページ、入力ページとは目に見える見えないにかかわらず、画面制御、テキスト画面、グラフィックの各ステートメントが実行されるページをいいます。

（１）ＷＩＤＴＨ　４０，２５，０またはＷＩＤＴＨ　４０，１２，０を設定しているとき、グラフィック画面は０～３ページの４画面、テキスト画面は０、１ページの２画面が使用できます。

パラメータｏとｉによるページ指定は次のようになります。

ｏ／ｉ	グラフィック画面のページ	テキスト画面のページ
0	0	0
1	1	1
2	2	0
3	3	1

（２）ＷＩＤＴＨ　８０，２５，０またはＷＩＤＴＨ　８０，１２，０を設定しているとき、グラフィック画面は０、１ページの２画面、テキスト画面は１画面使用できます。

<table>
<tr><th>o／i</th><th>グラフィック画面のページ</th><th>テキスト画面</th></tr>
<tr><td>0</td><td rowspan="2">0</td><td rowspan="4">1画面</td></tr>
<tr><td>1</td></tr>
<tr><td>2</td><td rowspan="2">1</td></tr>
<tr><td>3</td></tr>
</table>

（3）ＷＩＤＴＨ　４０，２５，１またはＷＩＤＴＨ　４０，１２，１を設定していているとき、グラフィック画面、テキスト画面共に２画面使用できます。

<table>
<tr><th>o／i</th><th>グラフィック画面のページ</th><th>テキスト画面のページ</th></tr>
<tr><td>0</td><td rowspan="2">0</td><td rowspan="2">0</td></tr>
<tr><td>2</td></tr>
<tr><td>1</td><td rowspan="2">1</td><td rowspan="2">1</td></tr>
<tr><td>3</td></tr>
</table>

（4）ＷＩＤＴＨ　８０，２５，１またはＷＩＤＴＨ　８０，１２，１を設定しているとき、グラフィック画面、テキスト画面共に１画面となります。

<table>
<tr><th>o／i</th><th>グラフィック画面</th><th>テキスト画面</th></tr>
<tr><td>0</td><td rowspan="4">1画面</td><td rowspan="4">1画面</td></tr>
<tr><td>1</td></tr>
<tr><td>2</td></tr>
<tr><td>3</td></tr>
</table>

〔II〕ＳＣＲＥＥＮまたはＧＲＡＰＨの第３パラメータｇは０～３の値をとり、アクセス可能なグラフィック画面を設定します。

すべてのグラフィックステートメントは、このパラメータにより設定されたグラフィック画面をアクセスします。

gの値	グラフィックモード
0	G1、G2、G3（全グラフィック画面）のアクセス可能
1	G1（初期状態が青の画面）のみアクセス可能
2	G2（初期状態が赤の画面）のみアクセス可能
3	G3（初期状態が緑の画面）のみアクセス可能

（1）g＝0のとき

グラフィックステートメントにより描かれているグラフィック画面はパレットコードに従います。

（2）g＝0以外のとき

グラフィックステートメント中のパレットコードが0以外のとき、そのグラフィック画面にドットをセットし、0のときはドットをリセットします。

2.6

パラメータを省略して

SCREEN ⏎

を実行した場合は、LIST⏎を実行したときと同様、グラフィック画面上の表示が消えて見えなくなります。この場合、パラメータを指定して実行すると、見えなくなる前のグラフィック画面が表示されます。

OPTION SCREEN1かOPTION SCREEN2を設定して、グラフィック1（&H4000／&HFFFF）すべてをVDIM変数エリアとして使用しているときは、入力ページの2ページと3ページを指定することはできません。

〈注　意〉　SCREENステートメントを実行して、画面の何も映らない状態になって困ったときは CTRL + D キーを押してください。

参　照　『ユーザーズマニュアル』の「ディスプレイモード」、2.8.1 OPTION SCREEN、2.6.2 WIDTH

サンプル
プログラム

```
10 ' SCREEN/GRAPH (例)
20 WIDTH 40,25
30 KMODE 1
40 SCREEN 0,0
50 PRINT"これは0ページ目です。"
60 SCREEN 0,1
70 PRINT"これは1ページ目です。"
80 SCREEN 0,0
90 A$=INKEY$(1)
100 SCREEN 1,1
110 A$=INKEY$(1)
120 GOTO 80
```

40、50：書き込むページを0に設定して、「これは0ページ目です」と書きます。

60、70：書き込むページを1に設定して、「これは1ページ目です」と書きます。

80：0ページを表示します。

100：1ページを表示します。

90、110：カーソルをブリンキングしてキーの入力があるまで待ちます。

2.6

2.6.2 WIDTH

機能 テキスト画面に表示する文字の数とグラフィック画面の解像度の設定をします。

書式

```
WIDTH c, l, g, d
```

c：1行当たりのけた数。４０または８０。
l：テキスト画面の行数。１０、１２、２０，２５の整数。
g：グラフィック画面の解像度の指定。０または１。
d：ディスプレイモードの指定。０、１、２の整数。

省略形 WI.

2.6

文例 WIDTH　４０，２５，０，１
⇨１行あたりのけた数４０、行数２５行、横３２０×たて２００ドットで標準ディスプレイモードに設定します。

解説

ＷＩＤＴＨは、テキスト画面に表示される文字の行数、けた数、グラフィック画面の解像度、および使用モニタの指定を行うためのステートメントです。

cに４０を指定すると、画面の１行の文字数は４０けた（３２０ドット）に設定され、８０を指定すると、８０けた（６４０ドット）に設定されます。

lは、テキスト画面の行数を設定するパラメータで、１０行、１２行、２０行、および２５行の指定をすることができます。

lによってテキスト画面を１０行か２０行に設定した場合、グラフィック画面を使用することはできません。また１０行の指定は標準ディスプレイモードの場合のみ使用可能です。

グラフィック画面の解像度の指定はgで行ない、第２パラメータlの値との組み合せにより、次のような指定をすることができます。

lの値	gの値	グラフィック画面	備　　考
１２	０	１９２(ライン)	標準と高解像度の両ディスプレイモードで設定可能
２５		２００	
１２	１	３８４	高解像度ディスプレイモードのみ設定可能
２５		４００	

dは、ディスプレイモードを設定するもので、次のような設定となります。

dの値	ディスプレイモードの設定
0	本体の標準／高解像度切換スイッチの状態に従う。 (▁)標準ディスプレイモード　(█)高解像度ディスプレイモード
1	標準ディスプレイモード
2	高解像度ディスプレイモード

このステートメントを実行すると、画面のサイズを設定すると同時に、

SCREEN　0，0　（出力、入力ページを0ページにする）
CLS4　（グラフィック、テキストの両画面をきれいに消す）

のステートメントが自動的に実行されます。ただし、OPTION　SCREENステートメントによって、グラフィック画面をグラフィックメモリとして使用している場合は画面の消去は行ないません。

詳しくは『ユーザーズマニュアル』の「ディスプレイモード」を参照にしてください。

2.6

〈注　意〉

標準ディスプレイでは、標準ディスプレイモードでのみ、高解度ディスプレイでは、高解度ディスプレイモードでのみ使用可能です。
また、標準ディスプレイモードを設定すると、gの値が1であっても0となります。

参　照

『ユーザーズマニュアル』の「ディスプレイモード」、
2.8.1 OPTION　SCREEN、2.6.1 SCREEN

サンプル
プログラム

```
10 ' WIDTH (例)
20 SCREEN0,0:KMODE 1
30 X=40:Y=12:GOSUB"ﾒｯｾｰｼﾞ"
50 X=40:Y=25:GOSUB"ﾒｯｾｰｼﾞ"
70 X=80:Y=12:GOSUB"ﾒｯｾｰｼﾞ"
90 X=80:Y=25:GOSUB"ﾒｯｾｰｼﾞ"
110 CLS4
120 END
1000 LABEL"ﾒｯｾｰｼﾞ"
1010 WIDTH X,Y
1020 Z=Y*8:IF Y=12 THEN Z=Z*2
1030 PRINT
1040 PRINT USING " WIDTH ##_,##_,9",X,Y
1050 PRINT
1060 PRINT USING " ﾃｷｽﾄ画面         ##×###
     #### 文字",X,Y,X*Y
1070 PRINT
1080 PRINT USING " ｸﾞﾗﾌｨｯｸ画面   ###×###
  ###### ﾄﾞｯﾄ",X*8,Z,X*8*Z
1090 PRINT
1100 PRINT USING "(高解像度ﾓﾆﾀ   ###×###
  ###### ﾄﾞｯﾄ)",X*8,Z*2,X*16*Z
1110 PRINT
1120 LINE(0,0)-(X*8-1,Z-1),PSET,1,B
1130 K$=INKEY$(1)
1140 RETURN
```

2.6

１０００～１１４０：画面の状態を説明するサブルーチン。

１０１０：画面の解像度の設定をします。

１０２０：Ｚはたてのライン数を表します。

１１３０：キーの入力があるまで待ち続けます。

2.6.3 CLS

機　能

画面を消去します。

書　式

```
〔1〕CLS
〔2〕CLS n
```

n：消去する画面の指定。0～4の整数。

文　例

CLS　4

⇨テキストは画面と3枚のグラフィック画面をすべて消去します。

解　説

CLSは、画面に表示されている文字やグラフィックを消すためのステートメントです。nの値によって、次のような画面の消去ができます。

nの値	消去する画面
省略	テキスト画面を消去
0	G1、G2、G3　すべてのグラフィック画面を消去
1	G1（初期状態が青の画面）を消去
2	G2（初期状態が赤の画面）を消去
3	G3（初期状態が緑の画面）を消去
4	すべてのグラフィック画面とテキスト画面を消去

nの値を省略して、テキスト画面を消去する場合、CONSOLEステートメントで画面の範囲が設定されているときは、その画面の中だけ消去します。テキスト画面全体を消去したいときは、CONSOLEを解除してからCLSを実行します。CONSOLEを解除するには、単独でCONSOLE⏎

グラフィック画面を消去する場合、WINDOWステートメントで画面の範囲が設定されているときは、その画面の中だけ消去します。グラフィック画面全体を消去したいときは、WINDOWを解除してからCLSを実行します。

WINDOWを解除するには、単独でWINDOW⏎を実行してください。

参　照

2.7.1 CONSOLE、2.8.2 WINDOW、
『ユーザーズマニュアル』の「ディスプレイモード」

サンプル
プログラム

```
10 ' CLS (例)
20 WIDTH40,25,0:SCREEN 0,0,0
100 FOR N=0 TO 4
110 GOSUB"カラーセット"
120 PRINT CHR$(12);" CLS";N;
130 K$=INKEY$(1)
140 CLS N
150 PRINT:PRINT"実行結果"
160 K$=INKEY$(1)
170 NEXT N
180 CLS 4
190 PRINT"終わり"
200 END
1000 LABEL"カラーセット"
1010 FOR I=1 TO7
1020 LINE(I*40,30)-(I*40+38,90),PSET,I,BF
1030 NEXT
1040 RETURN
```

グラフィック画面上に青から白までの７色を表示しこれをＣＬＳステートメントを使用して消去します。
ＣＬＳ Ｎ（Ｎは０～４）を表示しキー入力待ちになります。何かキー入力するとＣＬＳステートメントを実行し、「実行結果」と表示しキー入力待ちになります。何かキーをおすと次にＣＬＳの表示にもどります。
以上のサイクルをＣＬＳ０からＣＬＳ４までくりかえし、最後に「終わり」と表示し、終了します。

2.6.4 COLOR

機 能　表示画面の色を設定します。

書 式

```
COLOR 〔c1〕〔, c2〕
```

c_1：文字の表示色。カラーコード。0～7の整数。
c_2：画面の背景色。カラーコード。0～7の整数。

0：　黒。
1：　青。
2：　赤。
3：　マゼンタ（紫）。
4：　緑。
5：　シアン（水色）。
6：　黄。
7：　白。

2.6

省略形　COL.

文 例　COLOR　4，0
⇨文字の色（表示色）を緑、景背色を黒に設定します。

解 説

テキスト画面の文字の色と背景の色は、COLORステートメントで指定します。
（例1）COLOR　7，1……青地に白の文字が表示されます。
（例2）COLOR　1，5……シアン地に青の文字が表示されます。
グラフィック画面において、パレットコードを省略すると、COLORステートメントの表示色c1の値が、パレットコードの値として指定されます。
（例）210　COLOR　1
　　　220　PSET（100，100）……画面に青い点が表示されます。
BASIC起動時は、自動的にCOLOR　7，0に設定されています。
なお、表示色と背景色は、CTRLキーと数字キーを使って設定することができます。

参 照　『ユーザーズマニュアル』の「ディスプレイモード」、2.6.5 PALET

2.6.5 PALET／PALET@

機　　能　パレットコードの色とボーダカラーの色を設定します。

書　　式

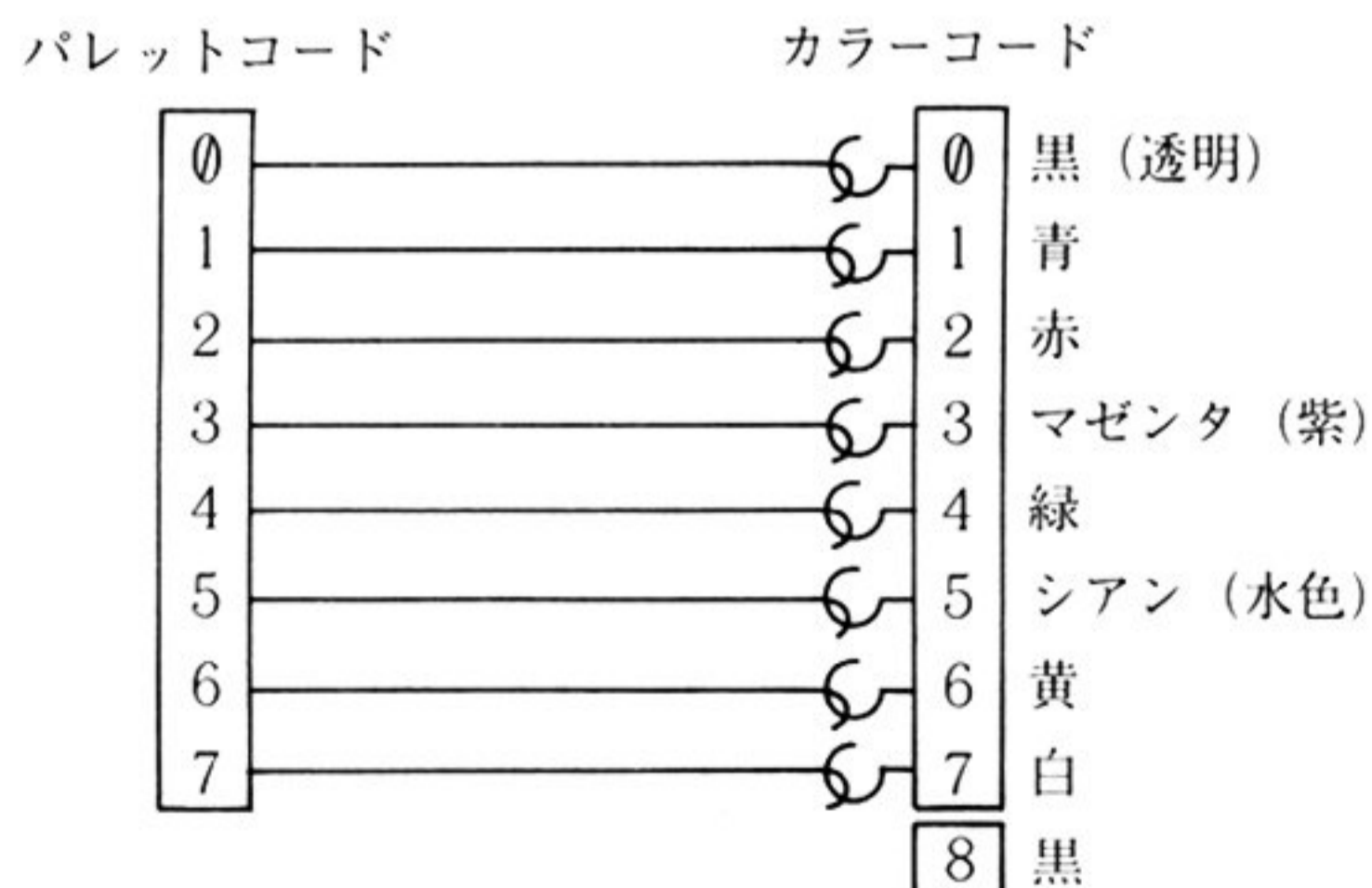
〔1〕PALET〔p, c, b〕
〔2〕PALET@ c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8

p：パレットコード。（0～7の整数）。
c：カラーコード。（0～7の整数）と8（ただし、8はパレットコード0およびパレットコード1としか指定できません）
b：0または1の整数（ボーダーカラーの指定です）
c_1～c_8：カラーコード。0～7の整数。

省 略 形　〔1〕PAL.　〔2〕PAL.@

2.6

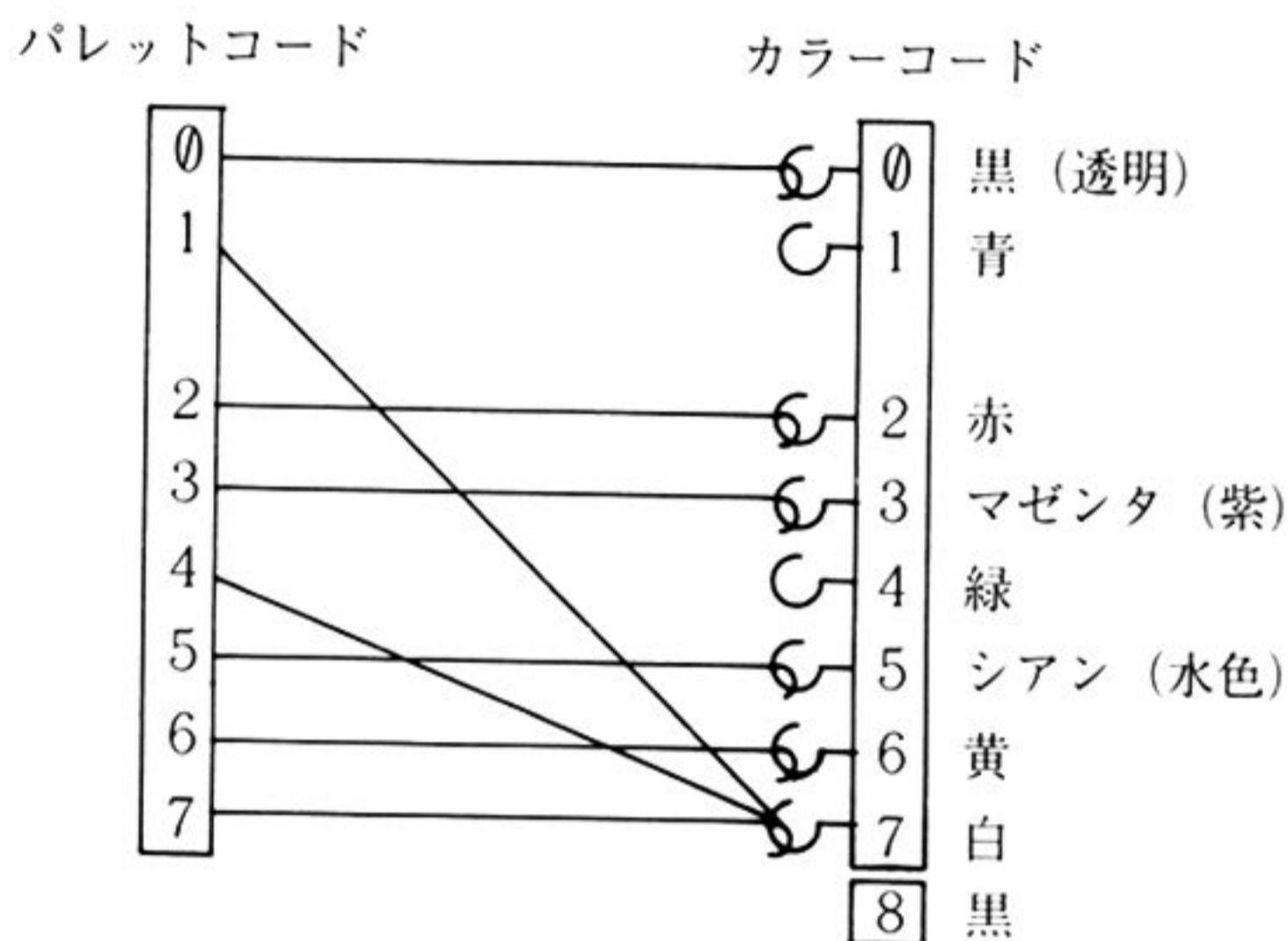
文　　例　PALFT 1，4，0
⇨パレットコード1を緑（カラーコード4）に切り換えます。ボーダーカラーはそのままとします。
PALET@1，1，2，3，4，5，6，6
⇨パレットコード0を青（カラーコード1）にし、パレットコード7を黄（カラーコード6）に指定します。

解　　説　パレットコードは色を指定する数字ですが、カラーコードのように数字に対する色が決まっていません。そこで、PALETステートメントを使って、パレットコードに対する色を決める必要があります。
下の図はPALETステートメントの概念を示すものです。

パレットコードとカラーコードはゴムひもでつながっていて、パレットコード側は固定され、カラーコード側にはつけはずしできるフックがついています。
BASIC起動時には、上の図のように、パレットコードがカラーコードにぴったり一致しています。したがって、パレットコードをカラーコードと同じ

ものとして使うことができます。
しかし、ここで、

PALET 1，7

PALET 4，7

というステートメントを実行すると、下の図のようにフックがかかって、パレットコードの1と4を白のコードとして使うことができます。

フックのつけ換えは、瞬時に行なうことができるので、プログラムで、

LINE（0，0）－（200，100），PSET，1，BF

と青いBOXを書いておき、次に

PALET 1，7

を実行すると、一瞬に青いBOXを白いBOXに変えることができます。

bは、ボーダーカラーの指定で1のとき黒抜き、0のとき初期状態となります。（省略すると0を指定したのと同じです。）黒色の実行は、スーパーインポーズ時でないとわかりません。また専用ディスプレイテレビが必要で、本体と専用ディスプレイテレビのVIDEO CUT端子をケーブルで接続しなければいけません。

（例）PALET 0，8，0

グラフィックのパレットコード0を黒色にします。ボーダーカラーはそのまま。

PALET 1，8，1

グラフィックのパレットコード1を黒色にします。ボーダーカラーも黒色にします。

単に、PALETを実行すると、BASIC起動時の初期状態に戻ります。

カラーコード0と8は通常は、両方とも黒となりますがスーパーインポーズ時には、8が黒色、カラーコード0が透明となります。

PALET@のパラメータの順序は、右から順にパレットコード 0、1、2、3、4、5、6、7となります。

参照　2.6.4 COLOR、2.6.6 PRW、『ユーザーズマニュアル』の「ディスプレイモード」

サンプルプログラム

プログラムを実行すると「ＰＡＬＥＴ」と表示させてパレットコードとカラーコードの入力待ちになるので、パレットコード、カラーコードの順に区切って入力してください。

ＰＡＬＥＴの実験をすることができます。

```
10 ' PALET (例)
20 INIT:WIDTH 40,25,0:KMODE 1
30 PRINT"パレットコード"
40 CREV 1
50 FOR I=0 TO 7
60 COLOR 7-I:LOCATE I*4,2:PRINT I;
70 LINE(I*32,24)-(I*32+23,127),PSET,I,BF

80 NEXT
90 LOCATE 28,2:COLOR 7:PRINT 7
100 CREV
110 LOCATE 0,16
120 PRINT SCRN$(0,2,30)
130 PRINT:PRINT"カラーコード"
140 LOCATE 0,19:PRINT CHR$(26)
150 C=9:INPUT"PALET ",P,C
160 IF C<8 THEN 190
170 INPUT"COLOR ",C
180 IF C>7 OR P>7 THEN INIT:END
190 LOCATE P*4,16:PRINTC
200 PALET P,C
210 GOTO140
```

2.6.6 PRW

機　能

文字（テキスト画面）とグラフィック図形（グラフィック画面）のどちらを優先して表示するかを指示します。

書　式

```
PRW〔n〕
```

n：0～255の整数。&B00000000～&B11111111または&H00～&HFF。

文　例

PRW　&B00010110　または　PRW22

⇨文字がグラフィックの青、赤、および緑の下に隠れます。

解　説

PRWは、文字とグラフィック図形が画面上で重なったとき、どちらかを上にして表示するかを決めるためのステートメントです。

具体的にどちらを上に表示するかは、0～7のパレットコードが示すそれぞれの色ごとに決められます。

nの値はコンピューター内部では8ビットで表現されており、各ビットは次のようにパレットコードに対応しています。

n	7	6	5	4	3	2	1	0	8ビットデータ
	白	黄	シアン	緑	マゼンタ	赤	青	黒	

※各色は、初期状態におけるパレットコードの色を示しています。

nの値によって、上の図の各ビットが1になったり、0になったりしますが、1のとき、その色のグラフィックの部分が文字の上に表示され（グラフィックの優先)、0のときそのグラフィック部分の上に表示されます。(テキストの優先)

（例）PRW　254……254は2進数で11111110なので、パレットコードが0の色以外のところでは、文字が隠れてしまい、まるでグラフィックの後ろに文字があるように見えます。この状態では、黒いグラフィック以外のところで文字が隠れて見えます。

PRWを実行して、カーソルが見えなくなったときは、CTRL+Dを押すと復帰することができます。

参　照

2.6.5 PALET、『ユーザーズマニュアル』の「ディスプレイモード」

サンプル
プログラム

```
10 ' PRW (例)
20 INIT:WIDTH 40,25,0:KMODE 1
30 FOR I=0 TO 7
40 LINE(I*32+8,24)-(I*32+31,95),PSET,7-I
,BF
50 NEXT
60 FOR I=1 TO 7
70 COLOR I:LINE(0,I+3)-(33,I+3),"A"
80 NEXT
90 FOR I=1 TO 7
100 B=2^I
110 B$=BIN$(B)
120 B$=RIGHT$("00000000"+B$,8)
130 LOCATE 0,1
140 PRINT"PRW &B";B$;"  or  PRW";B
150 PRW B
160 PAUSE 15
170 NEXT I
180 CLS 4
```

２０：画面の初期設定。

３０～５０：８色の長方形を描きます。

６０～８０：文字Ａを７行７色で描きます。

９０～１７０：ＰＲＷを実行するループ。

１５０：ＰＲＷを実行します。

2.6

2.6.7 CANVAS

機　　能	マルチページモードのとき、各グラフィック画面の色を設定します。

書　　式

```
CANVAS c1, c2, c3
```

c_1：G１の色を指定するカラーコード。0～7の整数。
c_2：G２の色を指定するカラーコード。0～7の整数。
c_3：G３の色を指定するカラーコード。0～7の整数。

省 略 形　　CAN.

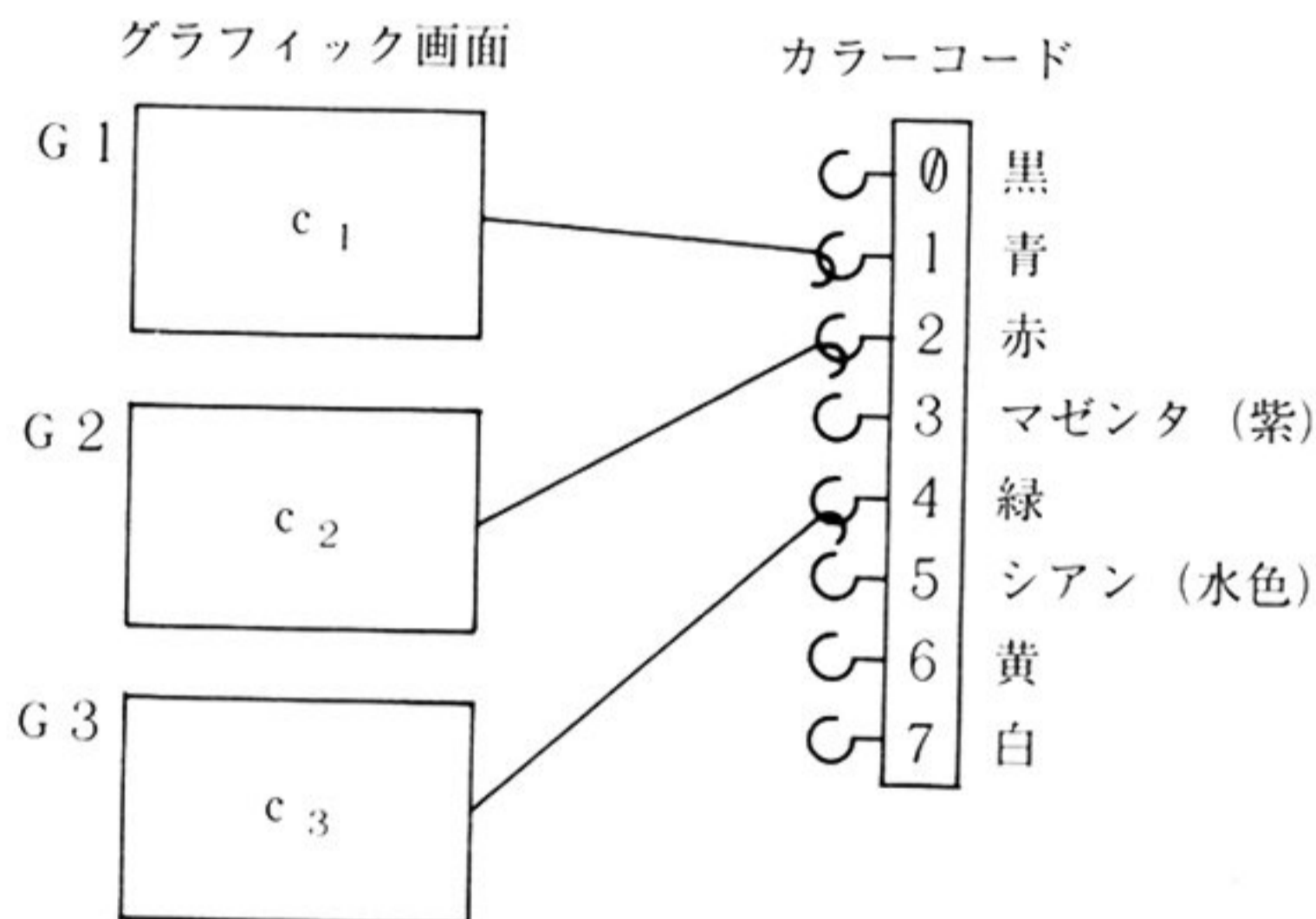
文　　例　　CANVAS　7，2，4

⇨グラフィック画面１を白、グラフィック画面２を赤、グラフィック画面３を緑にします。

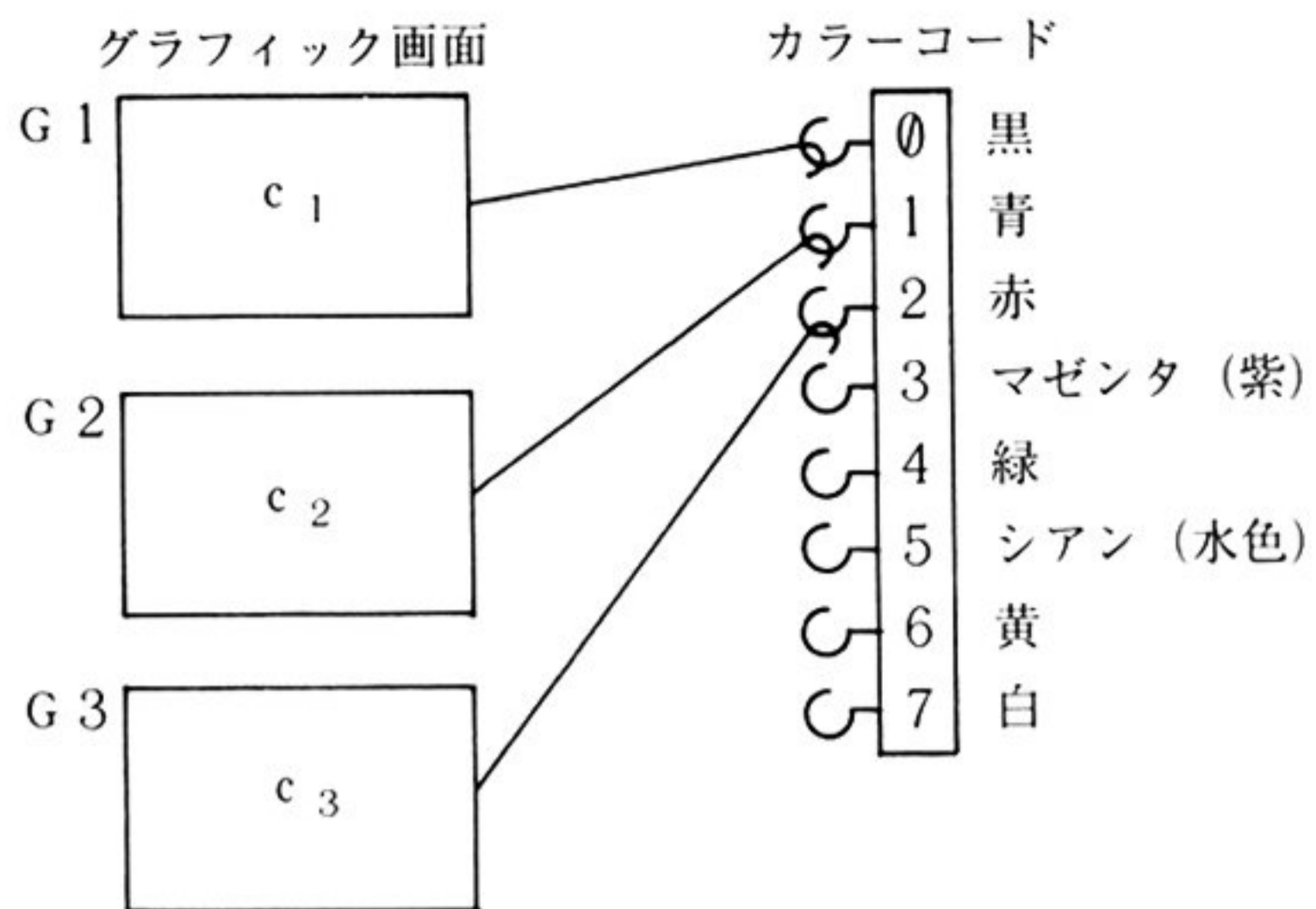
解　　説

グラフィック画面はG１、G２、G３の３枚あり、初期状態ではそれぞれ、G１が青、G２が赤やG３が緑に設定されています。

マルチページモードのとき、CANVASを使って、G１、G２、G３の各画面の色を変更することができます。

下の図はCANVASステートメントの概念を示すものです。

グラフィック画面とカラーコードはゴムひもでつながっていて、グラフィック画面側は固定され、カラーコード側にはつけはずしできるフックがついています。

BASIC起動時には、上の図のように、G１が1、G２が2、G３が4に設定されています。したがって、G１が青、G２が赤、G３が緑になっています。

しかし、ここで、CANVAS　0，1，2というステートメントを実行すると、次の図のようにフックがかかって、G１を黒、G２を青、G３を赤にすることができます。

ＣＡＮＶＡＳステートメントをＰＡＬＥＴやＰＲＷステートメントといっしょに使用しないでください。

2.6

参　　照　　2.6.8 ＬＡＹＥＲ

2.6.8 LAYER

機　能

テキスト画面と３つのグラフィック画面を重ねて表示するときの上下関係を指定します。

書　式

```
LAYER t, g1, g2, g3
```

t：テキスト画面の順位。１～４の整数。

g_1：Ｇ１（グラフィック画面１）の順位。１～４の整数。

g_2：Ｇ２（グラフィック画面２）の順位。１～４の整数。

g_3：Ｇ３（グラフィック画面３）の順位。１～４の整数。

文　例

ＬＡＹＥＲ　１，４，２，３

⇨１番上がテキスト画面、次がグラフィック画面Ｇ２、次がグラフィック画面Ｇ３、１番下がグラフィック画面Ｇ１となります。風景画にたとえれば、遠景から順にグラフィック画面Ｇ１、グラフィック画面Ｇ３、グラフィック画面Ｇ２、最も近景がテキスト画面となります。

解　説

ＬＡＹＥＲは、テキスト画面と３つのグラフィック画面が重なったとき、どういう順番で重ねて表示するかを決めるためのステートメントです。

t，g_1，g_2，g_3には、それぞれ１～４の数が入り、画面には順位の高いものが上になって表示されます。

１～４の数は、重複して指定することはできません。

ＬＡＹＥＲは、ＰＡＬＥＴやＰＲＷステートメントといっしょに使わないでください。

指定によってテキスト画面が見えず困ったときは［CTRL］＋［D］キーを押します。

参　照

2.6.7 ＣＡＮＶＡＳ、『ユーザーズマニュアル』の「ディスプレイモード」

2.6.9 KSEN

機　　能

アンダーラインを引いたり、アンダーラインの色の設定を行ないます。

書　　式

```
〔1〕KSEN〔m〕
〔2〕KSEN  m, c
```

m：アンダーラインを引くか否かの設定。0または1。

c：アンダーラインの色。カラーコード（0～7の整数）または8。

文　　例

KSEN　1，2

⇨赤いアンダーラインを引きます。

解　　説

ＫＳＥＮは、アンダーラインを引くか否かの設定と、アンダーラインの色を設定するためのステートメントです。

m＝0のとき、アンダーラインを引かず、m＝1のとき、アンダーラインを引くように設定することができます。

cに0～7のカラーコードを指定して、アンダーラインの色を設定することができます。

この命令が使えるのは、ＷＩＤＴＨによって10または20行のテキスト画面が設定されているときで、他の画面に対して、

　　ＫＳＥＮ　m，c

を使用すると、

　　ＰＡＬＥＴ　1，c

と同じ働きをしますので注意してください。

なお、mを省略するとＫＳＥＮ0と同じになります。

参　　照

2.6.2 ＷＩＤＴＨ　2.6.5 ＰＡＬＥＴ　2.9.3 ＨＣＯＰＹ

サンプル プログラム

これは、"パソコンテレビ"の所にアンダーラインを引くプログラムです。

```
10 ' KSEN <例>
20 INIT:WIDTH40,20:CLS4
30 PRINT"私は";
40 KSEN1,2:PRINT"パソコンテレビ";
50 KSEN:PRINT"です。"
```

2.6.10 GET@

機　　能　**グラフィック画面またはテキスト画面上に表示されている文字や図形のドットパターンを配列変数に読み込みます。**

書　　式

```
〔1〕GET@(x1, y1) − (x2, y2), A, c
〔2〕GET@(Xa, Ya) − (Xb, Yb), B
〔3〕GET@A (Xa, Ya) − (Xb, Yb), B
```

x_1，y_1：グラフィック画面の座標（画面座標系）
x_2，y_2：グラフィック画面の座標（画面座標系）

x_1，x_2：ＷＩＤＴＨ４０のとき、０～３１９の整数。
ＷＩＤＴＨ８０のとき、０～６３９の整数。

y_1，y_2：テキスト１２行のとき、０～１９１の整数。
または　０～３８３の整数。
テキスト２５行のとき、０～１９９の整数。
または　０～３９９の整数。

Ｘａ，Ｙａ：テキスト画面の座標
Ｘｂ，Ｙｂ：テキスト画面の座標

Ｘａ，Ｘｂ：ＷＩＤＴＨ　４０のとき、０～３９の整数。
ＷＩＤＴＨ　８０のとき、０～７９の整数。

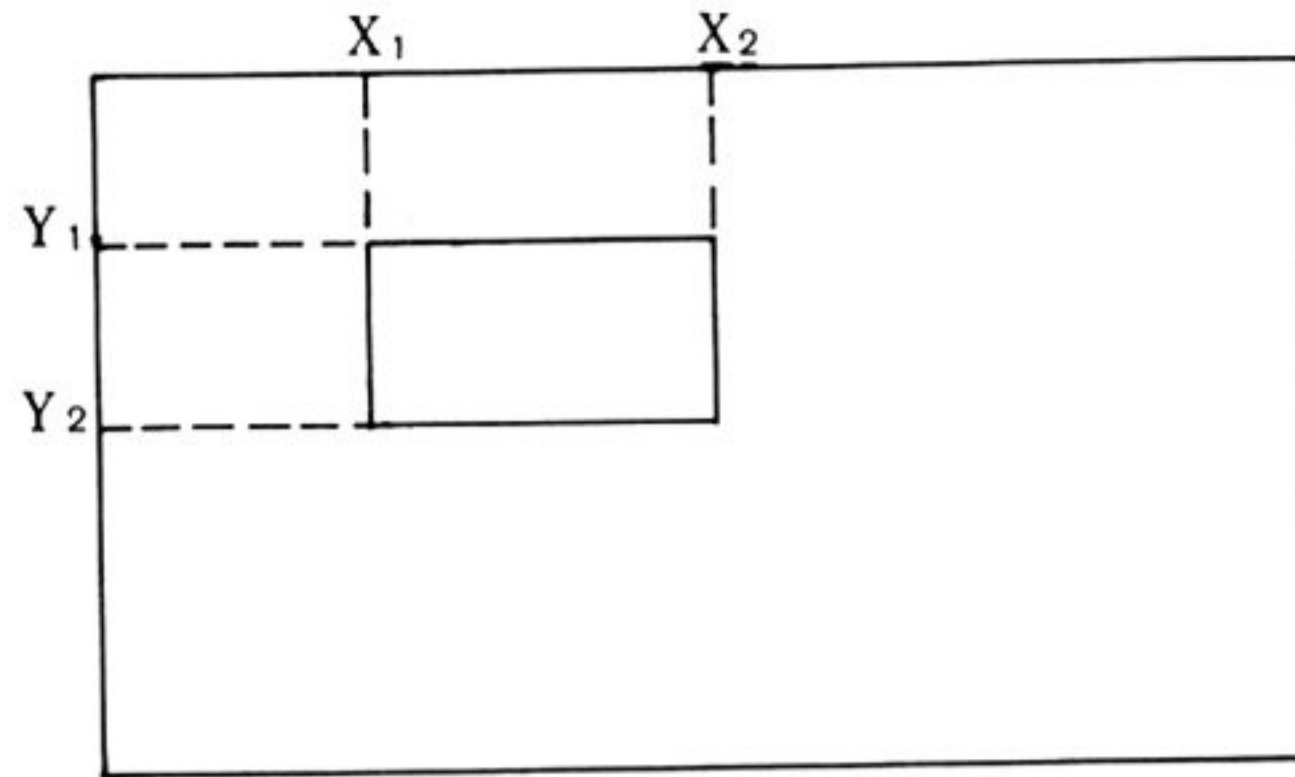
Ｙａ，Ｙｂ：テキスト１０行のとき、０～９の整数。
テキスト１２行のとき、０～１１の整数。
テキスト２０行のとき、０～１９の整数。
テキスト２５行のとき、０～２４の整数。

Ａ，Ｂ：配列名。ただし、配列変数名であること。プログラム中ですでに配列宣言してあること。（ＶＤＩＭ変数は使えません。）

ｃ：パレットコード。０～７の整数。

文　　例　ＧＥＴ＠（２０，２０）－（３０，３０），Ａ，７
⇨グラフィック画面上の（２０，２０）－（３０，３０）を対角とする長方形の枠内の青、赤、緑の画面のドットパターンを配列Ａに読み込みます。

解　　説　ＧＥＴ＠は、グラフィック画面上のドットパターンを配列に読み込むためのステートメントです。

読み込み領域は、（x_1，y_1）から（x_2，y_2）の座標または、（Ｘａ，Ｙａ）から（Ｘｂ，Ｙｂ）の座標を対角線とする長方形の枠の中です。

読み込むグラフィックの色は、c（パレットコード）で指定します。

cを省略すると、テキスト画面の文字を読み込むことができます。このとき、（Xa，Ya）と（Xb，Yb）の座標は、テキスト画面の文字の位置を示すことになります。

読み込んだデータを入れる配列はVDIM変数以外のものでなければなりません。

2.6

GET@Aでも、テキスト画面の文字を読み込みます。書式〔2〕との違いは漢字属性まで読み込みます。

グラフィック画面のドットパターンの読み込みは、グラフィックメモリ上の青の画面、赤の画面、緑の画面の3画面分のデータから、必要な画面の部分だけ順に読み込んで行われます。したがって、グラフィックの色によって読み込む画面の数が変わるので、読み込むドットの数もそれによって異なり、次のように計算されます。

黒のとき、	（読み込むドットの数）×0
青のとき、	（読み込むドットの数）×1（青の画面）
赤のとき、	（読み込むドットの数）×1（赤の画面）
マゼンタのとき、	（読み込むドットの数）×2（青、赤の画面）
緑のとき、	（読み込むドットの数）×1（緑の画面）
シアンのとき、	（読み込むドットの数）×2（青、緑の画面）
黄のとき、	（読み込むドットの数）×2（赤、緑の画面）
白のとき、	（読み込むドットの数）×3（青、赤、緑の画面）

配列にドットパターンを読み込むときは8ドットを1バイトのデータとして扱い、整数型のとき2バイト、単精度型のとき5バイト、倍精度型のとき8バイト必要なため、配列のサイズは、読み込む全ドット数を8で割り、さらに各型に必要なバイト数で割って、端数を切り上げたものとなります。

配列のサイズの具体的な計算式を次に示します。

＜グラフィック画面からドットパターンを読み込む場合＞

INT（((ABS（X1－X2）＋1）＊（ABS（Y1－Y2）＋1）＋7）／8）＊m／t

mの値は、パレットコードcによって異なります。

c＝0のとき、　m＝0
c＝1、2、4のとき、m＝1
c＝3、5、6のとき、m＝2
c＝7のとき、　m＝3

tの値は、変数名Aの型によって異なります。

整数型のとき、　　　t＝2

単精度型のとき、　　t＝5

倍精度型のとき、　　t＝8

<テキスト画面から文字を読み込む場合>

（ABS（x_1－x_2）＋1）＊（ABS（y_1－y_2）＋1）＊n／t

書式〔1〕〔3〕はn＝3、書式〔2〕はn＝2とします。

※nをかけるのは、文字といっしょにその属性や漢字属性も読み込むからです。

テキスト画面とその属性については、『ユーザーズマニュアル』の付録「テキスト画面（V－RAM）へのアクセス」を参照してください。

参　　照

2.6.11 PUT@、『ユーザーズマニュアル』の付録「テキスト画面（V－RAM）へのアクセス」

サンプルプログラム

画面の中央に描かれた円の一部（白枠で囲まれた部分）を配列A％に読み込み、左上に表示するプログラムです。

```
10 'GET@ (例)
20 CLS:WIDTH 40,25:PALET
30 OPTION BASE 1
40 DIM A%(84)
50 CIRCLE(159,99),40,1
60 PAINT(159,99),&H24,1
70 GET@(119,79)-(139,99),A%,7
80 LINE(118,78)-(140,100),PSET,7,B
90 PUT@(50,50)-(70,70),A%,PSET,7
100 END
```

2.6.11 PUT@

機　　能　ＧＥＴ@で読み込んだ文字や図形のドットパターンをグラフィック画面またはテキスト画面上に表示します。

書　　式

```
〔1〕PUT@(x1, y1) - (x2, y2), A, mode, c
〔2〕PUT@(Xa, Ya) - (Xb, Yb), B
〔3〕PUT@A (Xa, Ya) - (Xb, Yb), B
```

x_1，y_1：グラフィック画面の座標（画面座標系）
x_2，y_2：グラフィック画面の座標（画面座標系）

x_1，x_2：ＷＩＤＴＨ　４０のとき、０～３１９の整数。
ＷＩＤＴＨ　８０のとき、０～６３９の整数。

y_1，y_2：テキスト　１２行のとき、０～１９１の整数。
または、０～３８３の整数。
テキスト　２５行のとき、０～１９９の整数。
または、０～３９９の整数。

Ｘａ，Ｙａ：テキスト画面の座標
Ｘｂ，Ｙｂ：テキスト画面の座標

Ｘａ，Ｙａ：ＷＩＤＴＨ　４０のとき、０～３９の整数。
ＷＩＤＴＨ　８０のとき、０～７９の整数。

Ｘｂ，Ｙｂ：テキスト１０行のとき、０～９の整数。
テキスト１２行のとき、０～１１の整数。
テキスト２０行のとき、０～１９の整数。
テキスト２５行のとき、０～２４の整数。

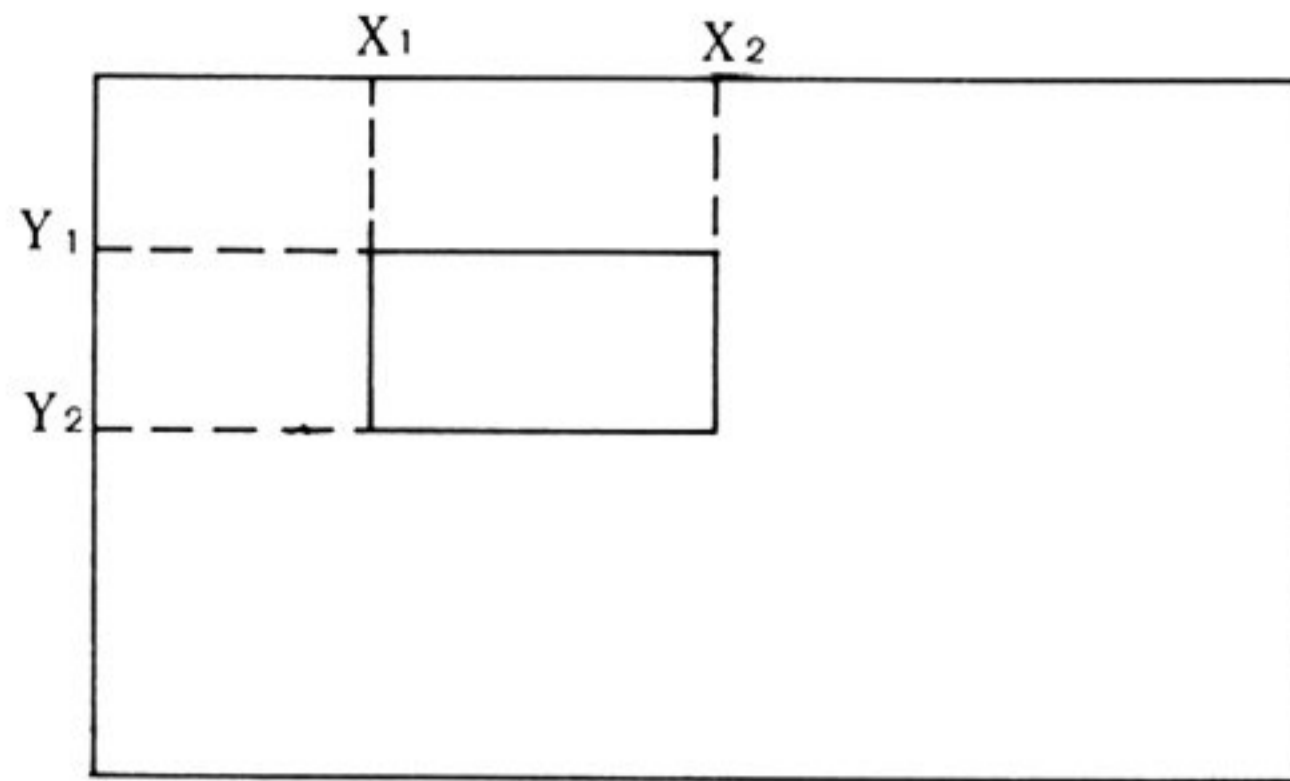
Ａ，Ｂ：配列名。ただし、配列変数名であること。プログラム中ですでに配列宣言してあること。（ＶＤＩＭ変数は使えません。）

ｍｏｄｅ：表示のモード。次の６つがあります。
ＰＳＥＴ、ＰＲＥＳＥＴ、ＸＯＲ、ＯＲ、ＡＮＤ、ＮＯＴ

ｃ：パレットコード。０～７の整数。

文　　例　ＰＵＴ@(２０，２０) − (３０，３０)，Ａ，ＰＳＥＴ，７
⇨画面上の（２０，２０）と（３０，３０）を対角とする長方形の領域内に、配列変数Ａにあるドットデータの青、赤、緑の画面の部分をグラフィック画面に表示します。

解　　説　ＰＵＴ@は、ＧＥＴ@で配列に読み込んだドットパターンを、画面の任意の位置に表示するためのステートメントです。

表示領域は、(x_1，y_1) と (x_2，y_2) または（Ｘａ，Ｙａ）と（Ｘｂ，Ｙｂ）を対角とする長方形の枠内です。

モードは、表示領域内にすでにあるドットと配列中のドットについて、ブール演算[1]を行ない、その結果を画面にセットするもので、次の6つがあります。

1）1.9.3論理演算子中のブール演算参照。

(1) ＰＳＥＴ：配列内のドットパターンをそのまま表示します。

(2) ＰＲＥＳＥＴ：配列内のビット1のドットを消します。

(3) ＸＯＲ：画面のドットとデータのドットをＸＯＲした結果を表示します。

(4) ＯＲ：画面のドットとデータのドットをＯＲした結果を表示します。

(5) ＡＮＤ：画面のドットとデータのドットをＡＮＤした結果を表示します。

(6) ＮＯＴ：配列内のドットパターンを反転して表示します。

・ＸＯＲとは、画面ドットとデータのドットを比較して違っている場合つまり、画面ドットが1でデータのドットが0であった時などに1となり表示するものです。

・ＯＲとは、比較して、両方とも0であった場合にのみ、0となり他は1となり表示するものです。

・ＡＮＤとは、比較して、両方とも1であった場合にのみ、1となり表示するものです。

テキスト画面のデータが入った配列のときには、モードとパレット・コードの指定はいりません。

注）書式〔1〕は、ＧＥＴ@の書式〔1〕と、書式〔2〕はＧＥＴ@の書式〔2〕と、書式〔3〕はＧＥＴ@の書式〔3〕と組み合わせて使ってください。

参　照　2.6.10 ＧＥＴ@

サンプルプログラム

画面中央に描かれた正六角形の一部分（白枠部分）を配列Ａ％に読み込み、左上に表示するプログラムです。

```
10 'PUT@ (例)
20 CLS:WIDTH 40,25:PALET
30 OPTION BASE 1
40 DIM A%(84)
50 POLY(159,99),40,4,60,0,360
60 PAINT(159,99),&H36,4
70 GET@(119,79)-(139,99),A%,7
80 LINE(118,78)-(140,100),PSET,7,B
90 PUT@(50,50)-(70,70),A%,PSET,7
100 END
```

2.7 テキスト画面ステートメント

2.7.1 CONSOLE

機　能　テキスト画面上での文字の表示エリアを設定します。

書　式

```
CONSOLE Ys ，YL [Xs ，XL]
```

Y_S ：たて方向の表示開始行。
　　１０行のとき、０～９。
　　１２行のとき、０～１１。
　　２０行のとき、０～１９。
　　２５行のとき、０～２４。

Y_L ：たて方向の表示行数。
　　１０行のとき、１～１０（正確には１～１０－Y_S）。
　　１２行のとき、１～１２（正確には１～１２－Y_S）。
　　２０行のとき、１～２０（正確には１～２０－Y_S）。
　　２５行のとき、１～２５（正確には１～２５－Y_S）。

X_S ：横方向の表示開始けた。
　　ＷＩＤＴＨ　４０のとき、０～３９。
　　ＷＩＤＴＨ　８０のとき、０～７９。

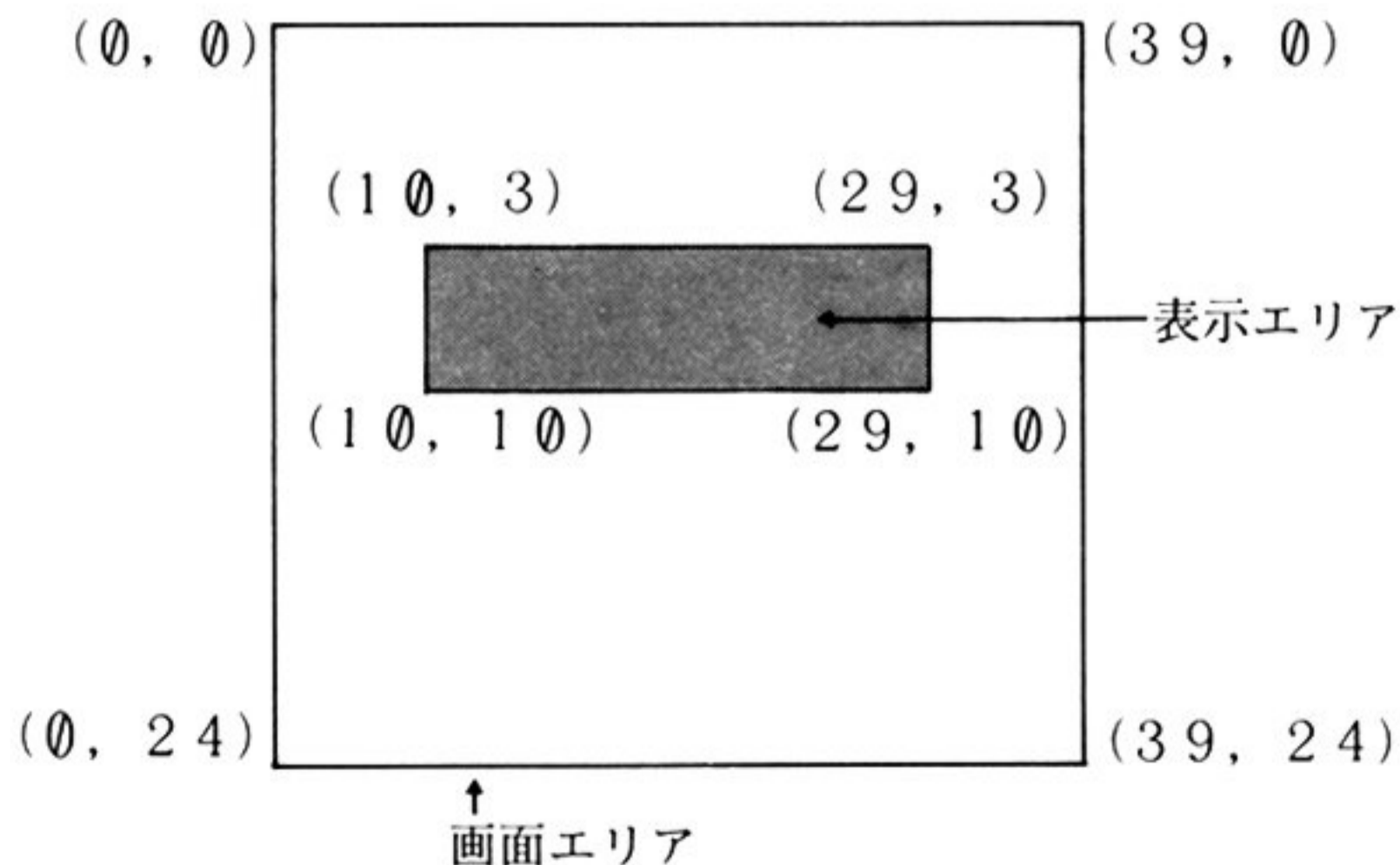
X_L ：横方向の表示けた数。
　　ＷＩＤＴＨ　４０のとき、１～４０。（正確には１～４０－X_S）。
　　ＷＩＤＴＨ　８０のとき、１～８０。（正確には１～８０－X_S）。

省略形　ＣＯＮＳ.

文　例　ＣＯＮＳＯＬＥ　３，８，１０，２０

⇨テキスト画面の文字の表示エリアが、（１０，３）、（２９，３）、（２９，１０）、（１０，１０）の長方形の中となります。

ＷＩＤＴＨ　４０，２５の場合下図の様になります。

解　説

テキスト画面に対して、文字の表示エリアを設定します。

このステートメントの実行後は、指定した長方形のエリア内だけに文字を表示することができ、テキスト画面のクリアやスクロールもこのエリア内で行なわれます。

設定後、カーソルは、(X_S, Y_S) の位置へ移動します。

ＣＯＮＳＯＬＥの後ろの指定をすべて省略すると、ＫＬＩＳＴ０およびＫＭＯＤＥ０ではテキスト画面全体を、その他ではファンクション表示部（最下行）を除くテキスト画面に戻ります。

なお、ＢＡＣＩＣ起動時、高解像度ディスプレイモードではＣＯＮＳＯＬＥ０，２４，０，８０、標準ディスプレイモードではＣＯＮＳＯＬＥ０，１１，０，８０が初期設定されています。

〈注　意〉

ＣＯＮＳＯＬＥのパラメータは表示エリアの始めと終りの座標を設定するのではないことに注意してください。

最下行のファンクションの表示は、ＣＯＮＳＯＬＥ０，２５：ＣＬＳで消去できますがＷＩＤＴＨステートメントを実行するとファンクション内容が表示されます。ファンクション表示部まで表示エリアに指示すると日本語入力モードになりません。

ファンクション表示を消すにはＫＬＩＳＴ０とし、ファンクション表示をするものはＫＬＩＳＴ１と指定してください。

2.7

参　照

『ユーザーズマニュアル』の「テキスト」

サンプルプログラム

```
10 ' CONSOLE (例)
20 INIT:WIDTH40,25,0:CLS4
30 FOR M=0 TO 1
40 KMODE M
50 LOCATE M*17+3,0
60 FOR I=0 TO 15
70 PRINT HEX$(I);
80 NEXT
90 CONSOLE 2,22,M*37,2
100 FOR I=0 TO 15
110 PRINT HEX$(I);"×";
120 NEXT
130 CONSOLE 2,22,M*17+3,16
140 FOR I=0 TO 255
150 PRINT#0,CHR$(I);
160 NEXT I
170 PRINT
180 PRINT"  KMODE";M
190 CONSOLE
200 NEXT
```

ＫＭＯＤＥ０とＫＭＯＤＥ１の場合のアスキーコード表をＣＯＮＳＯＬＥステートメントにより表示エリアをわけて表示するプログラム。

2.7.2 LOCATE／CURSOR

機　能　カーソルをテキスト画面の指定位置へ移動します。

書　式

```
[1] LOCATE x, y
[2] CURSOR x, y
```

x, y：CONSOLEステートメントで指定された表示エリア内の値。

　x：横方向の表示けた。WIDTH　4⌀のとき、⌀～39の整数。
　　　　　　　　　　　WIDTH　8⌀のとき、⌀～79の整数。

　y：たて方向の表示行。1⌀行のとき、⌀～9の整数。
　　　　　　　　　　　12行のとき、⌀～11の整数。
　　　　　　　　　　　2⌀行のとき、⌀～19の整数。
　　　　　　　　　　　25行のとき、⌀～24の整数。

省略形

[1] LOC.
[2] CU.

2.7

文　例

LOCATE　5, 1⌀
⇨テキスト画面のカーソルを（5, 1⌀）の位置にします。

解　説

　LOCATEおよびCURSORは、テキスト画面のカーソル位置を指定します。
　範囲は、CONSOLEで指定された表示エリア内で、それを外れた場合は、「Illegal function call」のエラーが出ます。

参　照

2.7.1 CONSOLE

サンプルプログラム

X、Yによって指定された座標に、LOCATE　X、Yを表示します。

```
10 ' LOCATE/CURSOR (例)
20 WIDTH 40,25
30 FOR X=0 TO 10 STEP 2
40 Y=INT(X/2)
50 LOCATE X,Y:PRINT "LOCATE";X;",";Y
60 NEXT X
70 END
RUN
LOCATE 0 , 0
  LOCATE 2 , 1
    LOCATE 4 , 2
      LOCATE 6 , 3
        LOCATE 8 , 4
          LOCATE 10 , 5
Ok
```

2.7.3 KMODE

機能　**画面に表示する文字のタイプを設定します。**

書式

```
KMODE 〔m〕
```

m：表示する文字のタイプ。0または1の整数。

省略形　KM.

文例　KMODE 1
⇨全角文字（主に日本語文字）を表示する設定をします。

解説

KMODEは、表示する文字の指定を行なうステートメントです。
mの値は表示する文字のタイプを表し、次の指定をすることができます。

mの値	表示可能な文字
0 または省略	1バイトコード文字（半角文字）
1	2バイトコード文字（全角文字）と セミグラフィック文字を除く半角文字

1バイトコード文字の指定は0～255のキャラクターコードによって、2バイトコード文字の指定は4けたの16進数で表されるシフトJISコードによって行なうことができます。

KMODE 1では、主に日本語表示が中心になり、キャラクタゼネレータ（CG）のセミグラフィック文字のコードが、シフトJISコードの一部として使用するため、セミグラフィック文字の表示ができなくなります。

KMODE 0では、セミグラィック文字が使用可能になるかわりに、全角表示ができなくなります。

また、BASIC起動時にはKMODE1 が初期設定されています。

〈注意〉

KMODE 1であっても、CONSOLEステートメントで、WIDTHステートメントで指定された行数の最下行まで表示エリアと指定すると、日本語入力モードになりません。

参照　付録「コードの体系」、『ユーザーズマニュアル』の「テキスト」

2.7

サンプルプログラム

```
10 ' KMODE (例1)
20 INIT:WIDTH40,25,0:CLS4
30 M=0:GOSUB"表"
40 M=1:GOSUB"表"
50 END
60 LABEL"表"
70 KMODE M
80 LOCATE M*17+3,0
90 FOR I=0 TO 15
100 PRINT HEX$(I);
110 NEXT
120 CONSOLE 2,22,A*37,2
130 FOR I=0 TO 15
140 PRINT HEX$(I);"x";
150 NEXT
160 CONSOLE 2,22,M*17+3,16
170 FOR I=0 TO 255
180 PRINT#0,CHR$(I);
190 NEXT I
200 PRINT
210 PRINT"  KMODE";M
220 CONSOLE
230 RETURN
```

ＫＭＯＤＥ０の場合と、ＫＮＯＤＥ１の場合のキャラクタジェネレータの文字を表示します。

2.7

2.7.4 CBLACK

機　能　**テキスト画面の文字の色を黒にします。**

書　式

```
CBLACK 〔n〕
```

n：カラーコード。0～7の整数。

文　例　CBLACK 1

⇨テキスト画面の青を黒にします。

解　説　テキスト画面に表示されている文字の色のうち、指定したカラーコードの色を黒に設定します。

カラーコードの0は、本来の黒色ではなく、透明色のため、たとえば、白の背景に黒い文字を表示するというようなことができません。そこで他のカラーコードを本当の黒色として、使えるようにしたのが、このステートメントです。ただし、カラーコード0自身を黒色にすることはできません。

nは、黒にしたいカラーコードで、次のような設定ができます。

nの値	
0 または省略	カラーコード　を透明色（標準の状態）にする。
1	青色を黒にする。
2	赤色を黒にする
3	マゼンタ（紫）を黒にする。
4	緑色を黒にする。
5	シアン（水色）を黒にする。
6	黄色を黒にする。
7	白色を黒にする。

参　照　2.6.4 COLOR

〈注　意〉　スーパーインポーズ時、テレビ画面に対して黒抜き表示を行なうには、専用ディスプレイテレビ（CZ－850DまたはCZ―855D）が必要です。（他のディスプレイテレビの場合は透明となります）ただし、本機の映像出力端子から画面出力を行なった場合は、どのモニターでも実行できます。また、高解像度ディスプレイモードではスーパーインポーズはできません。

2.7

サンプル
プログラム

```
10 ' CBLACK (例)
20 WIDTH40,12,0,0:INIT:KLIST0:CONSOLE0,12
30 A$="お誕生日おめでとう"
40 CSIZE3:CLS
50 COLOR1:CBLACK1:LOCATE10,1:PRINT"CBLACK1"
60 LINE(0,0)-(319,191),PSET,4,BF
70 LOCATE2,3:PRINT#0,A$;
80 LOCATE10,10:PRINT#0,"1985/04/30";
90 COLOR7
100 FOR I=0 TO 4000:NEXT
110 CBLACK0:LOCATE10,1:PRINT"CBLACK0"
120 LOCATE0,1:END
```

「お誕生日おめでとう」と「１９８５/０４/３０」がＣＢＬＡＣＫ１により黒文字で表示されます。すこしたつと黒文字がＣＢＬＡＣＫ０により青にかわります。これにつれてＣＢＬＡＣＫ１の表示がＣＢＬＡＣＫ０に変わります。

2.7

2.7.5 CREV

機　　能　**文字を反転させます。**

書　　式

```
CREV [m]
```

m：文字表示モード。0または1。
　0：標準モード。
　1：反転モード。

省 略 形　CR.

文　　例　CREV 1
⇨このステートメントの実行後、反転した文字が表示されます。

解　　説　CREVは、画面の文字を反転させるためのステートメントです。
（例1）CREV 1………文字を反転する（反転モードにする）。
（例2）CREV 0　あるいは単にCREV………文字をもとに戻す（標準モードにする）。

2.7

標準モードの文字。

Aaア1Ａａア１あ愛

反転モードの文字。

Aaア1Ａａア１あ愛

反転した文字は、次のように色そのものが反転した（カラーコードのビットが反転した）文字となります。ただし、地はすべて白になります。

（例）文字の色／地	反転文字の色／地
黒／黒………	白／白
青／黒………	黄／白
赤／黒………	シアン／白
マゼンタ／黒………	緑／白
緑／黒………	マゼンタ／白
シアン／黒………	赤／白
黄／黒………	青／白
白／黒………	黒／白

参　　照　『ユーザーズマニュアル』の「テキスト」

サンプルプログラム

"私はパソコンテレビです。"を7色で表示します。パソコンテレビの部分だけが、反転します。

```
10 ' CREV (例)
20 INIT:WIDTH40,25,0:KMODE1
30 FOR I=1 TO 7
40 COLOR I
50 PRINT"私は";
60 CREV 1
70 PRINT"パソコンテレビ";
80 CREV
90 PRINT"です。"
100 NEXT
```

2.7

2.7.6 CFLASH

機能　**文字を明滅させます。**

書式

```
CFLASH 〔m〕
```

m：文字表示モード。0または1。
　0：標準モード。
　1：明滅モード。

省略形　CF.

文例　CFLASH 1
⇨このステートメントの実行後、明滅する文字が表示されます。

解説　CFLASHは、画面の文字を明滅（ブリンキング）させるためのステートメントです。
（例1）CFLASH 1………文字を明滅する（明滅モードにする）。
（例2）CFLASH 0 あるいは単に CFLASH………文字をもとに戻す（標準モードにする）。
明滅モードとは、CREVステートメントの標準モードと反転モードとを交互に繰り返すモードで、文字が明滅しているように見えます。明滅の周期は、カーソルの明滅の周期に一致します。

参照　『ユーザーズマニュアル』の「テキスト」

2.7.7 CSIZE

機　能　**文字を拡大します。**

書　式

```
CSIZE〔n〕
```

n：文字拡大モード。0～3の整数。
　0：標準文字。
　1：たて2倍文字。
　2：横2倍文字。
　3：たて横2倍文字。

省略形　CS.

文　例　CSIZE　3
⇨たて横2倍された文字を表示します。

2.7

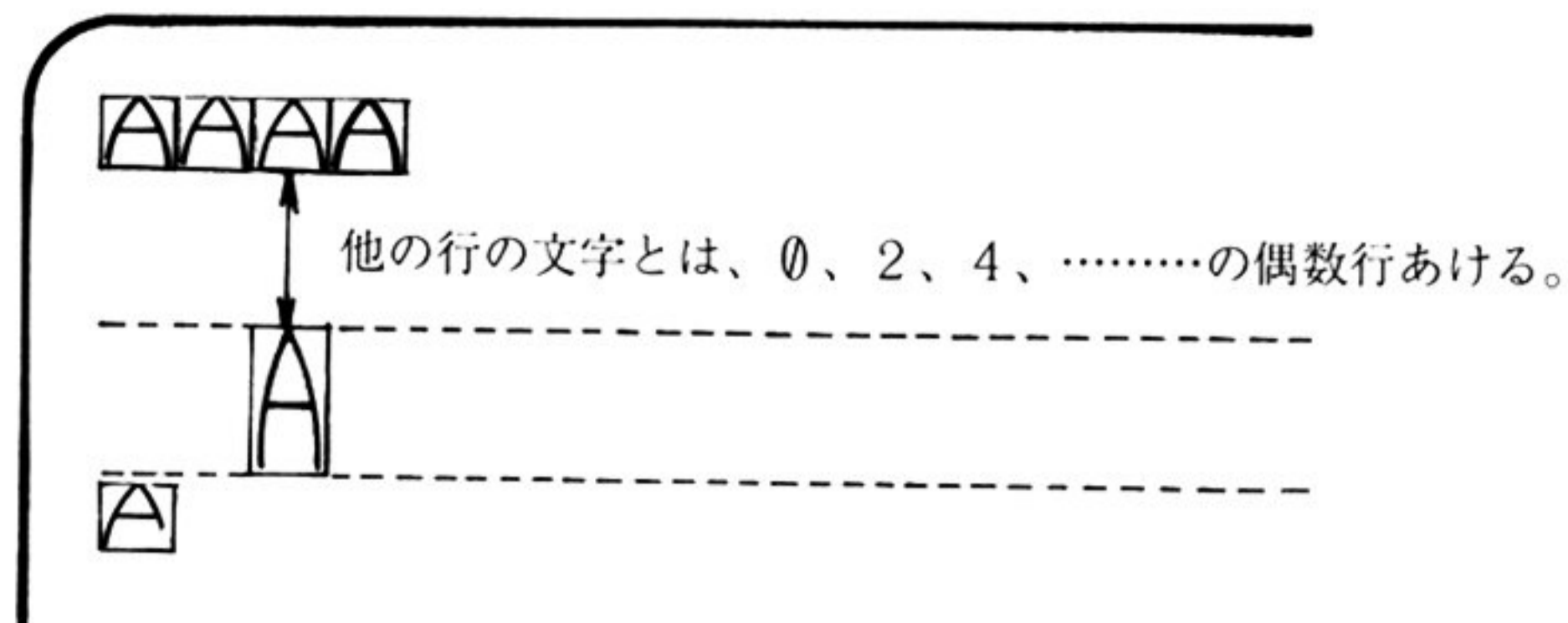
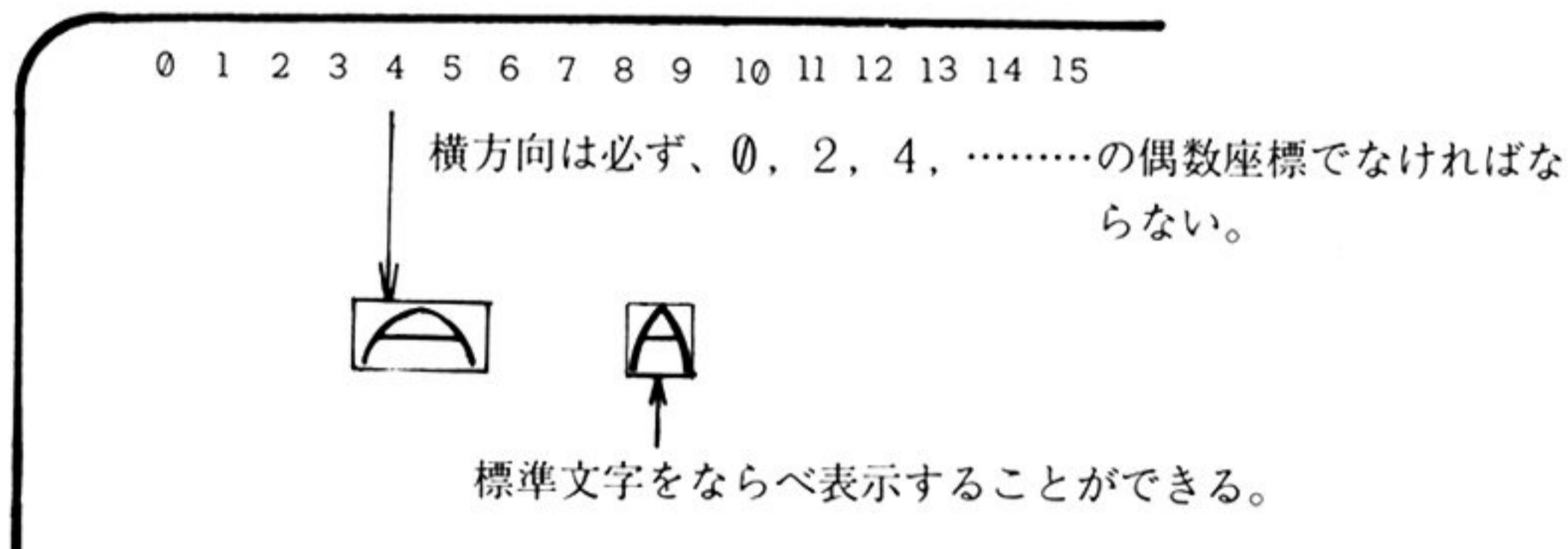
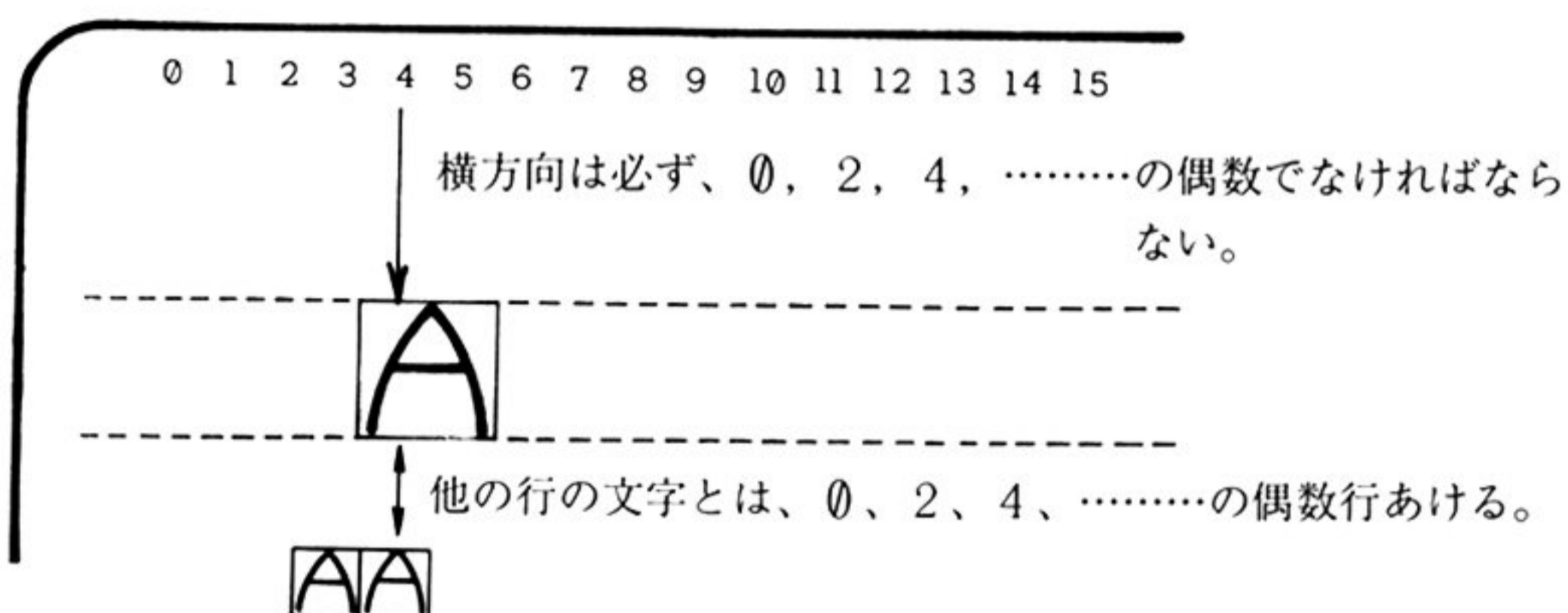
解　説　CSIZEは、文字のサイズを拡大するためのステートメントです。
（例1）CSIZE　0　あるいは単に　CSIZE………標準の大きさの文字を表示する。
（例2）CSIZE　1………たて2倍文字を表示します。
（例3）CSIZE　2………横2倍の文字を表示します。
（例4）CSIZE　3………たて横2倍文字を表示します。
LOCATEとCURSORによる座標の指定は、次のような斜線部に対するものとなります。
　たて2倍文字▨、横2倍文字▨□、たて横2倍文字▨□

　これらの拡大文字を表示するには、PRINT＃0ステートメントを使用しなければなりません。単なるPRINTステートメントを使うと、標準文字が表示されます。
　PRINT＃0で拡大文字を表示するときは、次の点に注意してください。

(1)　CSIZE　1（たて2倍文字）のとき
　・行の違う他の文字とは、0，2，4，………の偶数行あける。
　・その文字のある2行内に標準および横2倍文字があってはならない。

(2)　CSIZE　2　（横2倍文字）のとき
　・横方向は必ず、0，2，4，………の偶数座標でなければならない。
　・その文字のある行に標準文字を表示することができる。

(3)　CSIZE　3　（たて横2倍文字のとき）
　・横方向は必ず、0，2，4，………の偶数座標でなければならない。
　・その文字のある2行内に標準および横2倍文字があってはならない。
　・行の違う他の文字とは、0、2，4，………の偶数行あける。

・CSIZE　1（たて2倍文字）のとき

・CSIZE　2（横2倍文字のとき）

・CSIZE　3（たて横2倍文字）のとき

たて２０行または１０行の画面のときは、横2倍文字のみ表示することができます。なお、たて2倍文字、たて横2倍文字のみ表示することで、たて２０、２５行時、標準ディスプレイモードでも１６×8、１６×１６ドット文字の表示が行なえます。

参　照	2.3.7 PRINT＃0

サンプル
プログラム

それぞれのサイズの文字を表示します。

```
10 ' CSIZE(例)
20 INIT:WIDTH 40,25,0:KMODE 1
30 CSIZE 3
40 CLS:REM テキスト属性のクリア
50 CSIZE 0
60 LOCATE 0,0:PRINT"CSIZE 0 "
70 PRINT#0,"標準文字  ABC"
80 CSIZE 1
90 LOCATE 0,3:PRINT"CSIZE 1 "
100 PRINT#0,"たて２倍文字  ABC"
110 CSIZE 2
120 LOCATE 0,7:PRINT"CSIZE 2 "
130 PRINT#0,"横２倍文字  ABC"
140 CSIZE 3
150 LOCATE 0,10:PRINT"CSIZE 3 "
160 PRINT#0,"たて横２倍文字  ABC"
```

2.7.8 CGEN

機能　文字の表示モードの切り換えをします。

書式

```
CGEN〔m〕
```

m：文字の表示モード。0、1、2の整数。

0：標準モード（ROMCGモード）。

1：8×8ドットのユーザー定義キャラクタを表示するモード（RAMCGモード）。

2：16×8ドットのユーザー定義キャラクタを表示するモード（RAMCGモード）。

省略形　CG.

文例　CGEN　2

⇨16×8ドットのユーザー定義キャラクタを表示します。

解説

テキスト画面に表示される文字は、キャラクタゼネレータによって生成されたものです。

キャラクタ・ジェネレータには、ROMCG（Read Only Memory Character Generator）とRAMCG（Random Access Memory Character Generator）の2つがあります。ROMCGには、すでに256文字のドットパターン（アルファベットを始めとする図形文字）が記憶されており、RAMCGには、ユーザーが256文字まで自由にドットパターンを記憶することができるようになっています。

CGENは、ROMCGの文字を表示する（ROMCGモード）か、RAMCGの文字を表示する（RAMCGモード）かを設定するためのステートメントです。

（例1）　CGEN　2……RAMCGモードの設定をします。これによって、ユーザーの作った16×8ドットの文字を表示することができます。

（例2）　CGEN　1………RAMCGモードの設定をします。これによって、ユーザーの作った8×8ドットの文字を表示することができます。

（例3）　CGEN　0　または単に　CGEN………ROMCGモードの設定をします。

RAMCGモードにすると、それ以後表示される文字は、すべてユーザーがRAMCGに記憶させたドットパターンの文字となります。

ユーザーがRAMCGに文字のドットパターンを記憶させるには、DEFCHR$ステートメントを使います。

参照　2.7.9 DEFCHR$

サンプル
プログラム

```
10 ' CGEN (例)
20 INIT:WIDTH40,25,0:KMODE0
30 A$=""
40 FOR I=0 TO 7
50 READ D$
60 A$=A$+CHR$(VAL("&B"+D$))
70 NEXT
80 DEFCHR$(13)=A$
90 CGEN1
100 PRINT#0,CHR$(13)
110 CGEN
120 KMODE1
130 END
140 DATA 00000000 :'=&H00
150 DATA 00000010 :'=&H02
160 DATA 00000010 :'=&H02
170 DATA 00010010 :'=&H12
180 DATA 00100010 :'=&H22
190 DATA 01111100 :'=&H7C
200 DATA 00100000 :'=&H20
210 DATA 00010000 :'=&H10
```

ＣＨＲ＄（１３）に定義された「↲」表示がＣＧＥＮステートメントとにより表示されます。

2.7

2.7.9 DEFCHR$

機　　能　RAMCGに文字のドットパターンを記憶します。

書　　式

```
DEFCHR$ (n) =x$
```

n：コードを表わす数式。単独の定数、数値変数。
x＄：文字式。

文　　例　DEFCHR＄（&H8Ø）＝A＄

⇨ユーザ定義のキャラクタゼネレータ（RAMCG）のキャラクタコード&H8Øに対して文字変数A＄で与えられたパターンを定義します。

解　　説　DEFCHR＄は、RAMCGに文字のドットパターンを記憶させるためのステートメントです。

ユーザーがRAMCGに記憶させることのできる文字（ユーザー定義文字）は、たて×横＝8×8ドット、16×8ドット、16×16ドットの3つのタイプがあります。

ユーザー定義文字をRAMCGに記憶させるには、

8×8ドットのパターンで1文字に24バイト（青、赤、緑の順に8バイトずつ）
16×8ドットのパターンで1文字に48バイト（青、赤、緑の順に16バイトずつ）
16×16ドットのパターンで1文字に96バイト（青、赤、緑の順に32バイトずつ）。

の文字列が必要です。

ただし、8×8ドットのパターンで1文字8バイト
16×8ドットのパターンで1文字16バイト
16×16ドットのパターンで1文字32バイト

の文字列を設定すると、青、赤、緑のすべて同じ設定となり、表示色が白の文字の定義となります。

定義できるコードnは、キャラクタのサイズによって次のような範囲となっています。

- 8×8ドット　：Ø～255
- 16×8ドット　：&H1ØØ、&H1Ø2、……、&H1FE。
- 16×16ドット：&J7621～&J766ØのJIS16進定数

（例）　DEFCHR＄（32）＝"XXX……X"
　　　　（XXX……X：24バイト）

…… キャラクタコードの32に"XXX……X"で示されるドットパターンの文字を定義します。

（例）　DEFCHR＄（32）＝HEXCHR＄（"HHHH……HH"）
　　　　（HHHH……HH：16進数表現48けた）

…… キャラクタコードの３２にＨＥＸＣＨＲ＄で示されるドットパターンの文字を定義する。

ユーザー定義文字は、８×８ドット、１６×８ドット指定の場合、前もってＣＧＥＮステートメントを実行してＰＲＩＮＴ＃０ステートメントによって表示できますが、１６×１６ドット指定の場合、ＫＭＯＤＥ１を実行した後、普通のＰＲＩＮＴステートメントを使って表示できます。

(例)　１１００　ＣＧＥＮ１
　　　１１１０　ＰＲＩＮＴ＃０、ＣＨＲ＄（３２）
　　　…… キャラクタコード３２のユーザー定義文字を表示します。

参　考

『ユーザーズマニュアル』の付録「ユーザー定義文字の作り方」、2.7.8 ＣＧＥＮ

サンプルプログラム

(例１)

```
10 ' DEFCHR$ (例1)
20 INIT:WIDTH40,25,0:KMODE0
30 A$=""
40 FOR I=0 TO 7
50 READ D$
60 A$=A$+CHR$(VAL("&B"+D$))
70 NEXT
80 DEFCHR$(13)=A$
90 CGEN1
100 PRINT#0,CHR$(13)
110 CGEN
120 KMODE1
130 END
140 DATA 00000000 :'=&H00
150 DATA 00000010 :'=&H02
160 DATA 00000010 :'=&H02
170 DATA 00010010 :'=&H12
180 DATA 00100010 :'=&H22
190 DATA 01111100 :'=&H7C
200 DATA 00100000 :'=&H20
210 DATA 00010000 :'=&H10
```

↲の形のＤＡＴＡをＲＡＭＣＧに定義し、左上に表示します。

(例２)

```
10 ' DEFCHR$ (例2)
20 INIT:WIDTH40,25,0:KMODE0
30 A$=""
40 FOR I=0 TO 15
50 READ D$
60 A$=A$+CHR$(VAL("&B"+D$))
70 NEXT
80 DEFCHR$(&H1FE)=A$
90 CGEN2
100 PRINT#0,CHR$(&HFE)
110 CGEN
120 KMODE1
130 END
140 DATA 00000000 :'=&H00
150 DATA 00011000 :'=&H14
160 DATA 00111100 :'=&H3C
170 DATA 01111110 :'=&H7E
180 DATA 00011000 :'=&H18
190 DATA 00011000 :'=&H18
200 DATA 00011000 :'=&H18
210 DATA 00011000 :'=&H18
220 DATA 00011000 :'=&H18
230 DATA 00011000 :'=&H18
240 DATA 00011000 :'=&H18
250 DATA 00011000 :'=&H18
260 DATA 01111110 :'=&H7E
270 DATA 00111100 :'=&H3C
280 DATA 00011000 :'=&H18
290 DATA 00000000 :'=&H00
```

↕の形のＤＡＴＡをＲＡＭＣＧに定義し、左上に表示します。

2.7.10 POKE@

機能

VRAM内の指定されたアドレスに1バイトのデータを書き込みます。

書式

```
POKE@ a,m〔, n, ……〕
```

a：VRAM内のアドレス。&H2000～&HFFFF。
m，n，………：数式。0～255の整数を値にもつ式。

省略形

PO.@

文例

POKE@　&H4000，&HFF
⇨VRAM内のアドレス&H4000に1バイトのデータ&HFFを書き込みます。この場合、グラフィック画面のLINE（0，0）－（8，0），PSET，1と同様です。

解説

POKE@は、I／Oポート内のVRAMに1バイトのデータを書き込むステートメントです。

データをカンマ（,）で区切って書くと、連続したアドレスに書き込むことができます。

POKE@は、OUTステートメントとほとんど同じ動作をしますが、&H2000未満のアドレスに書き込もうとすると、「Illegal function call」のエラーを出すため、VRAM以外のポートに間違って書き込むことを防ぐことができます。

参照

2.3.12 POKE、2.3.13 OUT、『ユーザーズマニュアル』の付録「テキスト画面（V－RAM）へのアクセス」

サンプルプログラム

```
10 ' POKE@ (例)
20 WIDTH40.25.0:SCREEN0.0:CLS4
30 PRINT
40 DEFCHR$(&H53)=STRING$(3.HEXCHR$("0E10
203C04081060"))
50 POKE@ &H3000.&H53.&H48.&H41.&H52.&H50

60 POKE@ &H2000.&H22.&H13.&HC.&H5.&H6
70 LOCATE0.10
80 END
```

"SHARP"という文字を表示します。
Sは40行目のDEFCHR$で定義した文字。
Hは明滅した文字。
Aは反転した文字。
R，Pは，それぞれシアン，黄の文字となります。

2.8 グラフィックステートメント

2.8.1 OPTION SCREEN

機能

グラフィックメモリの使用目的を設定します。

書式

```
OPTION SCREEN n
```

n：設定モード。0～4の整数。

省略形

OP. SC.

文例

OPTION SCREEN 4
⇨2つのグラフィックメモリを外部記憶用（MEM0：～MEM1：）として使う設定をします。

解説

グラフィックメモリは、本来のグラフィック表示のためのみならず、データやプログラムを記録する普通のメモリとして使用することができます。

OPTION SCREENは、このグラフィックメモリを、グラフィック表示用に使用するか、外部記憶装置として使用するか、または変数エリアとして使用するか、の設定を行なうステートメントです。

グラフィックメモリは、グラフィックRAMとして2つ用意されており、1つ目をグラフィックメモリ0、2つ目をグラフィックメモリ1と呼んで区別します。グラフィックメモリの使用目的は、nの値によって、次のように設定されます。

n	グラフィックメモリ0	グラフィックメモリ1
0	グラフィック表示用	グラフィック表示用
1	グラフィック表示用	変数エリア
2	外部記憶装置用（MEM0：）	変数エリア用
3	グラフィック表示用	外部記憶装置用（MEM1：）
4	外部記憶装置用（MEM0：）	外部記憶装置用（MEM1：）

※（　）内は、デバイス名を示しています。

BASIC起動時は、n＝1の状態に設定されています。

グラフィックメモリを外部記憶装置用として使用する場合、指定するデバイス名は、グラフィックメモリ0のとき「MEM0：」、グラフィックメモリ1のとき「MEM1：」となります。

グラフィックメモリを外部記憶装置用として使用する場合は、あらかじめ初

期化を行なっておく必要があります。

（例1）１００　ＯＰＴＩＯＮ　ＳＣＲＥＥＮ　２
　　　　１１０　ＩＮＩＴ"ＭＥＭ０："………グラフィックメモリ０の初期化（フォーマットの管理テーブルの初期化）を行ないます。

（例2）１００　ＯＰＴＩＯＮ　ＳＣＲＥＥＮ　３
　　　　１１０　ＩＮＩＴ"ＭＥＭ１："………グラフィックメモリ１の初期化（フォーマットの管理テーブルの初期化）を行ないます。

〈注　意〉　ＷＩＤＴＨ４０，２５
ＯＰＴＩＯＮ　ＳＣＲＥＥＮ　０
の状態でグラフィック画面の入力ページが、２または３に設定されているとき、
ＯＰＴＩＯＮ　ＳＣＲＥＥＮ　１　またはＯＰＴＩＯＮ　ＳＣＲＥＥＮ　２
を実行すると、入力ページが２から０、３から１に切りかわります。

ＳＣＲＥＥＮ　２，２のとき　ＳＣＲＥＥＮ　２，０
ＳＣＲＥＥＮ　３，３のとき　ＳＣＲＥＥＮ　３，１

ただし、ＯＰＴＩＯＮ　ＳＣＲＥＥＮ　０の状態にする以前にＶＤＩＭ　ＣＬＥＡＲ　ステートメントでアドレスが指定されていたときは切りかわりません。
また、ＷＩＤＴＨ,，１，２の設定時にＯＰＴＩＯＮ　ＳＣＲＥＥＮ　１またはＯＰＴＩＯＮ　ＳＣＲＥＥＮ　２を実行すると「Ｂａｄ　ｓｃｒｅｅｎ　ｍｏｄｅ」のエラーとなります。

参　照　『ユーザーズマニュアル』の「フリーエリアについて」

2.8

2.8.2 WINDOW

機　能　グラフィックを、画面のどこに、どんなスケールで表示するのか設定します。

書　式

```
WINDOW (xs, ys) − (xe, ye) [, (x1, y1) − (x2, y2)]
```

xs，ys：画面表示の始点座標（画面座標系）
xe，ye：画面表示の終点座標（画面座標系）
　　xs，xe：WIDTH　40のとき、0～319。
　　　　　　WIDTH　80のとき、0～639。
　　ys，ye：12行のとき、0～191または0～383。
　　　　　　25行のとき、0～199または0～399。
x1，y1：始点座標（ユーザー座標系）
x2，y2：終点席標（ユーザー座標系）
　　約−1.7E+38～1.7E+38の実数。
　　（ただし、指数部は−38～+38）。

省略形　WIN.

2.8

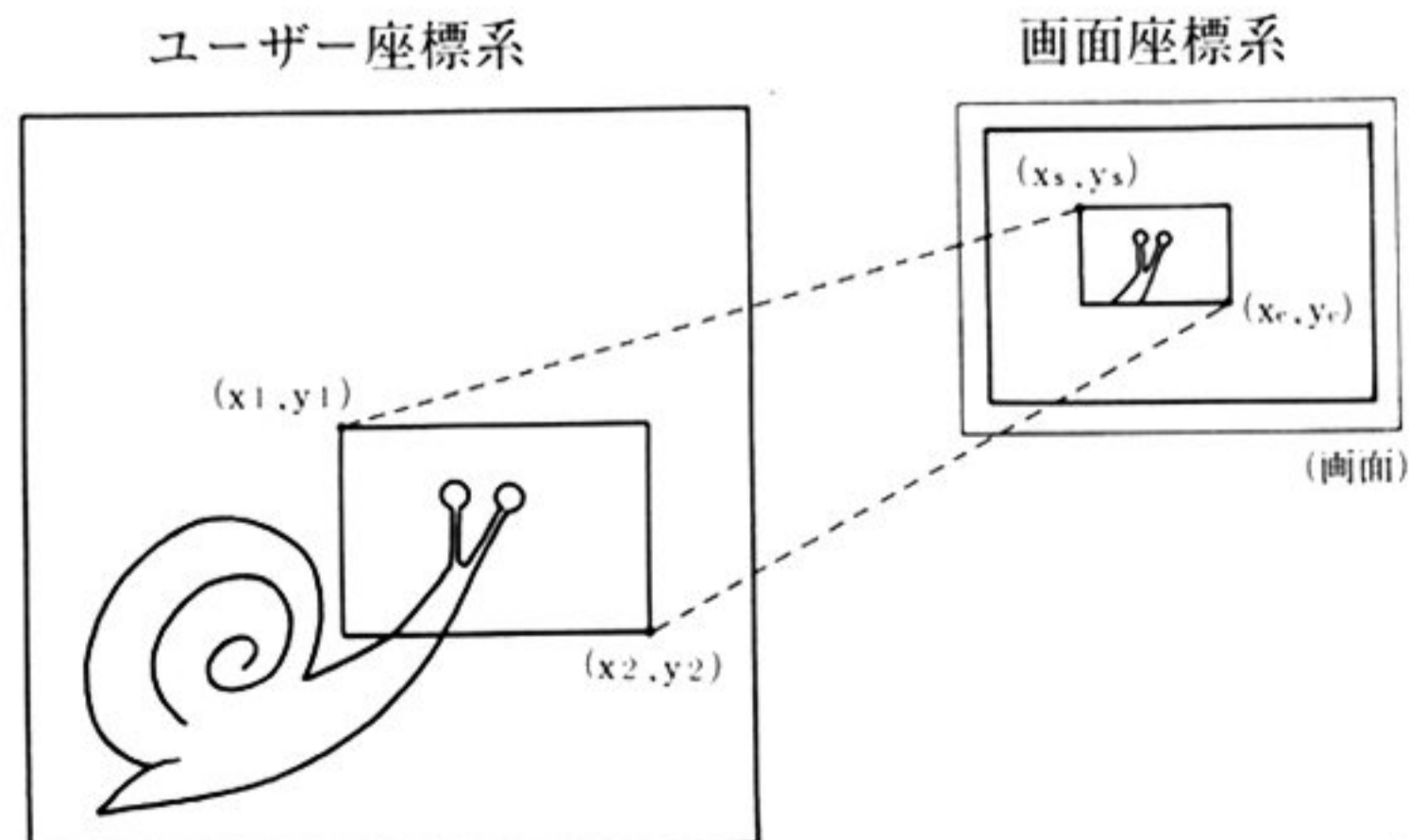
文　例　WINDOW (100, 50) − (300, 150), (−200, −200) − (200, 200)

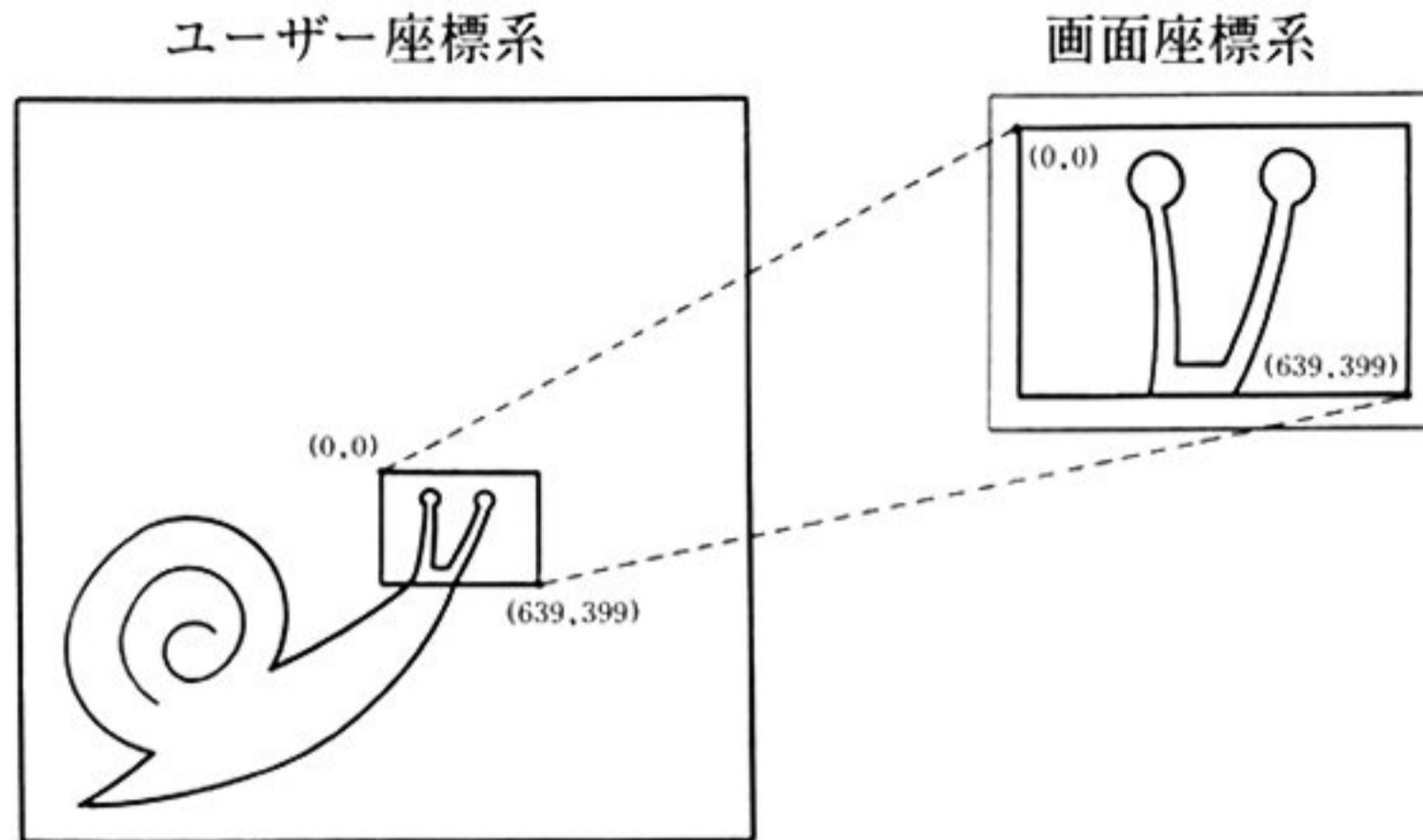
⇨(100,50)、(300,150)を対角とする長方形の画面座標を(−200, −200)、(200, 200) を対角とする長方形のユーザー座標とみなします。

解　説　WINDOWステートメントを使うと、ユーザー座標平面の中から任意の領域を指定して、画面上の任意の領域に表示させることができます。

たとえば、ユーザー座標系の（X1，Y1）から（X2，Y2）を対角線とする長方形の領域を画面上の（Xs，Ys）から（Xe，Ye）を対角線とする長方形の領域に表示させるには、

WINDOW (Xs, Ys) − (Xe, Ye), (X1, Y1) − (X2, Y2)

と指定します。次にその概念図を示します。

ユーザー座標系の範囲ならば、マクロ・ミクロのどんな領域でも、画面に自由にスケールを変えて表示することができます。つまり、拡大、縮少ができます。

ユーザー座標の指定を省略すると、次のように画面座標系に一致した領域が指定されます。

参　照　『ユーザーズマニュアル』の「ディスプレイモード」

サンプルプログラム　ＷＩＮＤＯＷステートメントによって設定されたスケールで星を表示します。（画面座標一定）

```
10 ' WINDOW (例)
20 INIT:WIDTH40,25,0:CLS0
30 C=7
40 GOSUB"星"
50 WINDOW(0,0)-(319,199),(0,0)-(639,160)

60 C=6
70 GOSUB"星"
80 WINDOW(0,0)-(319,199),(0,0)-(639,400)

90 C=5
100 GOSUB"星"
110 WINDOW(0,0)-(319,199),(0,0)-(639,160
0)
120 C=4
130 GOSUB"星"
140 END
150 LABEL"星"
160 POLY(160,100),50,C,144,90,810
170 RETURN
```

2.8

2.8.3 PSET

機　能　グラフィック画面に点（ドット）をセットします。

書　式

```
PSET (x,y [, c])
```

x，y：ドットの表示位置（WINDOWで指定したユーザー座標系）。BASIC起動時の座標は、次のように画面座標系と一致しています。
　x：横の座標。　WIDTH　40のとき、0～319の整数。
　　　　　　　　　WIDTH　80のとき、0～639の整数。
　y：たての座標。テキスト12行のとき、0～191の整数。
　　　　　　　　　　　　　　または、0～383の整数。
　　　　　　　　テキスト25行のとき、0～199の整数。
　　　　　　　　　　　　　　または、0～399の整数。

c：パレットコード。0～7の整数。

省略形　PS.

文　例　PSET（20，20，4）
⇨（20，20）にドットを緑色で表示します。

解　説　グラフィック画面上の座標（x，y）にドットを指定のパレットコードでセットします。
（x，y）が、WINDOWで設定した範囲の外のときは表示されません。
SCREENによってグラフィックコードが指定されているときは、G1、G2、G3のうちアクセス可能なグラフィック画面に対しドットを表示します。
CANVASとLAYERによってマルチページモードが指定されているときは、指定のグラフィック画面の色でドットを表示します。

参　照　2.8.4 PRESET

2.8.4 PRESET

機能　グラフィック画面の点（ドット）をリセットします。

書式

```
PRESET (x, y〔, c〕)
```

x，y ：ドットの表示位置（WINDOWで指定したユーザー座標系）。BASIC起動時の座標は次のように画面座標と一致しています。

x ：横の座標。　WIDTH　40のとき、0～319の整数。
　　　　　　　　WIDTH　80のとき、0～639の整数。

y ：たての座標。テキスト12行のとき、0～191の整数。
　　　　　　　　　　　または、0～383の整数。
　　　　　　　　テキスト25行のとき、0～199の整数。
　　　　　　　　　　　または、0～399の整数。

c：パレットコード。0～7の整数。

省略形　PRE.

文例　PRESET（20，20，4）

⇨（20，20）のドットを緑色でリセットします。たとえば、白で書かれた所では、白がマゼンダに変わります。

解説

グラフィック画面上の座標（x，y）のドットを指定のパレットコードでリセットします。

リセットするとは、各グラフィック画面に対しドットが表示されていれば消し、表示されていなければ何もしないことをいいます。

（x，y）が、WINDOWで設定した範囲の外のときは何もしません。

SCREENによってグラフィックモードが指定されているときは、G1、G2、G3のうちアクセス可能なグラフィック画面にドットをリセットします。

CANVASとLAYERによってマルチページモードが指定されているときは、指定のグラフィック画面のドットをリセットします。ただし、パレットコードの色が黒のときは見かけ上なにもしません。

参照　2.8.3 PSET

2.8.5 LINE

機　　能　直線を引きます。

書　　式

```
〔1〕LINE〔(x1, y1)〕−(x2, y2)〔, mode, c, ls〕
〔2〕LINE〔(x1, y1)〕−(x2, y2)〔, x$〕
```

x_1, y_1：直線の始点座標。
x_2, y_2：直線の終点座標。
どちらもＷＩＮＤＯＷで指定したユーザー座標系の座標です。ただし、ＢＡＳＩＣ起動時の座標は次のように画面座標と一致しています。

x_1, x_2：横方向の座標。　ＷＩＤＴＨ　４０のとき、０～３１９。
ＷＩＤＴＨ　８０のとき、０～６３９。
y_1, y_2：たて方向の座標。テキスト１２行のとき、０～１９１。
または、０～３８３。
テキスト２５行のとき、０～１９９。
または、０～３９９。

※文字式ｘ＄が指定されているときは、テキスト画面の座標となります。

x_1, x_1：横方向の座標。　ＷＩＤＴＨ　４０のとき、０～３９。
ＷＩＤＴＨ　８０のとき、０～７９。
y_2, y_2：たて方向の座標。テキスト１０行のとき、０～９。
テキスト１２行のとき、０～１１。
テキスト２０行のとき、０～１９。
テキスト２５行のとき、０～２４。

ｍｏｄｅ：ＰＳＥＴ，ＰＲＥＳＥＴ、ＸＯＲの３つがあります。
ｃ：パレットコード。０～７の整数。
ｌｓ：ラインスタイル。０～＆ＨＦＦＦＦ。
ｘ＄：文字式。単独の文字列、変数、配列、文字関数およびそれらを連結したもの。ここでは、文字列の先頭文字だけが意味をもちます。

文　　例

```
LINE (0, 0) − (100, 100), PSET, 4, &HCCCC
```

⇨（０，０）－（１００，１００）までパレットコード４の色で点線を引きます。

解　　説

このステートメントを実行すると、(x_1, y_1) を始点、(x_2, y_2) を終点とする直線を引きます。前に実行したＬＩＮＥステートメントの終点を始点とする場合は、(x_1, y_1) を省略することができます。

座標の後ろに指定するｍｏｄｅには、ＰＳＥＴ，ＰＲＥＳＥＴ、ＸＯＲの３つがあります。ＰＳＥＴは「直線を引く」、ＰＲＥＳＥＴは「直線を消す」、ＸＯＲは「直線の色を消す」の意味をもっています。ＸＯＲを指定すると、重なった部分の色が消されます。

ｍｏｄｅの後ろのパレットコードを省略すると、ＣＯＬＯＲやＰＡＬＥＴステー

トメントで指定している色で表示されます。

ラインスタイルlsは、点線、破線などを16ビットのビットパターンで表わしたものです。たとえば、&HFFCCという一点鎖線を引く場合のビットパターンは、一点鎖線が―――――――――　――

のパターンの繰り返しなので、

1111 1111 1100 1100
= F F C C

したがって、ラインスタイルは16進数表示で&HFFCC、2進数表示で&B1111111111001100となります。

ラインスタイルlsを省略すると&HFFFFの実線が引かれます。

次に主なラインスタイルとその破線のパターンを示します。

ラインスタイル	破線のパターン
&HFFFF	―――――――――――――― 実線
&HFF00	―― ―― ――
&HCCCC	― ― ― ― ― ―
&HAAAA	- - - - - - - - - - - - - - 点線
&HFFCC	――――――――― ―
&HE4E4	――― ― ――― - 一点鎖線
&HFCCC	―――――― ―――--
&HFF24	――――――― - - 二点鎖線

座標の後ろに文字式を書くと、テキスト画面に対して文字の直線を描くことができます。このとき、(x_1, y_1) から (x_2, y_2) を、文字式x$の表わす文字列の先頭の文字で結びつきます。ただし、WINDOWステートメントによって画面座標系とユーザー座標系を別の座標系としているときは、(x_1, y_1) と (x_2, y_2) はユーザー座標とみなされるので注意が必要です。

〈注　意〉　パレットコードcおよびラインスタイルlsを数値変換で指定するときは、先頭がBで始まる変数名を使うことができません。

参　照　LINE

サンプルプログラム　様々な色の直線を引きます。

80番の最後に、"&HCCCC"を追加すると点線になります。

80番のCの後ろに、"B"を追加するとBOXになります。

またBOXを点線にするときには、さらに"&HCCCC"を追加します。

```
10 ' LINE <例>
20 INIT:WIDTH40,25,0:CLS4
30 X1=INT(RND*320)
40 X2=INT(RND*320)
50 Y1=INT(RND*200)
60 Y2=INT(RND*200)
70 C=INT(RND*7)+1
80 LINE(X1,Y1)-(X2,Y2),PSET,C
90 K$=INKEY$
100 IF K$=" " THEN INIT:CLS4:END
110 GOTO30
```

2.8

機能 長方形を描きます。

書式

```
〔1〕LINE〔(x1, y1)〕－(x2, y2), mode〔, c〕, B〔, ls〕
〔2〕LINE〔(x1, y1)〕－(x2, y2), x$, B
```

x_1，y_1：直線の始点座標。
x_2，y_2：直線の終点座標。
どちらもＷＩＮＤＯＷで指定したユーザー座標系の座標です。ただし、ＢＡＳＩＣ起動時の座標は次のように画面座標系と一致しています。

x_1，x_2：横方向の座標。ＷＩＤＴＨ ４０のとき、０～３１９。
ＷＩＤＴＨ ８０のとき、０～６３９。
y_1，y_2：たて方向の座標。テキスト１２行のとき、０～１９１。
または、０～３８３。
テキスト２５行のとき、０～１９９。
または、０～３９９。

※文字式ｘ＄が指定されているときは、テキスト画面の座標となります。

x_1，x_2：横方向の座標。ＷＩＤＴＨ ４０のとき、０～３９。
ＷＩＤＴＨ ８０のとき、０～７９。
y_1，y_2：たて方向の席標。テキスト１０行のとき、０～９。
テキスト１２行のとき、０～１１。
テキスト２０行のとき、０～１９。
テキスト２５行のとき、０～２４。

ｍｏｄｅ：ＰＳＥＴ，ＰＲＥＳＥＴ，ＸＯＲの３つがあります。
ｃ：パレットコード。０～７の整数。
Ｂ：長方形を描く指定。
ｌｓ：ラインスタイル。０～＆ＨＦＦＦＦ。
ｘ＄：文字式。単独の文字列、変数、配列、文字関数およびそれらを連結したもの。文字列の先頭の１文字のみ意味をもちます。

文例 ＬＩＮＥ（５０，５０）－（１５０，１５０），ＰＳＥＴ，４，Ｂ
⇨（５０，５０）と（１５０，１５０）を対角とする長方形をパレットコード４の色で描きます。

解説 このステートメントを実行すると、(x_1，y_1)と(x_2，y_2)の２点を対角とする長方形を描きます。(x_1，y_1)を省略すると、その前のＬＩＮＥの(x_2，y_2)から新しい(x_2，y_2)を対角とする長方形を描きます。

座標の後ろに指定するｍｏｄｅには、ＰＳＥＴ、ＰＲＥＳＥＴ、ＸＯＲの３つがあります。ＰＳＥＴは「直線を引く」、ＰＲＥＳＥＴは「直線を消す」、ＸＯＲは「直線の色を消す」の意味をもっています。ＸＯＲを指定すると、重なっ

た部分の色が消されます。

ｍｏｄｅの後ろのパレットコードを省略すると、ＣＯＬＯＲやＰＡＬＥＴステートメントで指定している色で表示されます。

ｃは、パレットコードを指定します。

Ｂは、長方形を描くために必要な指定です。

ラインスタイルｌｓは、点線、破線などを１６ビットのビットパターンで表わしたものです。詳しくは、前項のＬＩＮＥの説明を参照してください。

座標の後ろに文字式を書くと、テキスト画面に対して文字で長方形を描くことができます。このとき描かれる長方形は、（x_1，y_1）と（x_2，y_2）の両座標間を対角とする文字の枠です。ただし、ＷＩＮＤＯＷステートメントによって画面座標系とユーザー座標系を別の座標系としているときは、（x_1，y_1）と（x_2，y_2）はユーザー座標とみなされるので注意が必要です。

〈注　　意〉　パレットコードｃを数値変換で指定するときは、先頭がＢで始まる変数名を使うことができません。

参　　照　ＬＩＮＥ

2.8

機　能　長方形を描き、その中を色で塗りつぶします。

書　式

```
〔1〕LINE〔(x1, y1)〕−(x2, y2), mode, BF
〔2〕LINE〔(x1, y1)〕−(x2, y2), mode〔, c〕, BF
〔3〕LINE〔(x1, y1)〕−(x2, y2), mode, BF, t$
〔4〕LINE〔(x1, y1)〕−(x2, y2), x$, BF
```

x_1, y_1：直線の始点座標。
x_2, y_2：直線の終点座標。
どちらもＷＩＮＤＯＷで指定したユーザー座標系の座標です。ただし、ＢＡＳＩＣ起動時の座標は次のように画面座標系と一致しています。

x_1, x_2：横方向の座標。　ＷＩＤＴＨ　４０のとき、０～３１９。
ＷＩＤＴＨ　８０のとき、０～６３９。
y_1, y_2：たて方向の座標。テキスト１２行のとき、０～１９１。
または、０～３８３。
テキスト２５行のとき、０～１９９。
または、０～３９９。

※文字式ｘ＄が指定されているときは、テキスト画面の座標となります。

x_1, x_2：横方向の座標。　ＷＩＤＴＨ　４０のとき、０～３９。
ＷＩＤＴＨ　８０のとき、０～７９。
y_1, y_2：たて方向の座標。テキスト１０行のとき、０～９。
テキスト１２行のとき、０～１１。
テキスト２０行のとき、０～１９。
テキスト２５行のとき、０～２４。

ｍｏｄｅ：ＰＳＥＴ、ＰＲＥＳＥＴ、ＸＯＲの３つがあります。
ＢＦ：長方形の中を塗りつぶす（ボックスフィル）指定。
ｃ：パレットコード（０～７の整数）、または中間色（＆－Ｈ００～＆Ｈ７Ｆ）。
ｔ＄：タイリングパターン。
ｘ＄：文字式。単独の文字列、変数、配列、文字関係およびそれらを連結したもの。文字列の先頭の１文字のみ意味をもちます。

文　例　ＬＩＮＥ（５０，５０）－（１５０，１５０），ＰＳＥＴ，＆Ｈ２６，ＢＦ
⇨（５０，５０）と（１５０，１５０）を対角とする長方形の内部全体を赤と黄色の中間色で塗ります。

解　説　このステートメントを実行すると、(x_1, y_1) と (x_2, y_2) の２点を対角とする長方形を描いて中を指定のパレットコードｃかタイリングパターンｔ＄で塗りつぶします。(x_1, y_1) を省略すると、その前のＬＩＮＥの (x_2, y_2) から、新しい (x_2, y_2) を対角とする長方形を描いて、中を塗りつぶします。

座標の後ろに指定するｍｏｄｅには、ＰＳＥＴ、ＰＲＥＳＥＴ、ＸＯＲの３つがあります。ＰＳＥＴは「直線を引く」、ＰＲＥＳＥＴは「直線を消す」、ＸＯＲは「直線の色を消す」の意味をもっています。ＸＯＲを指定すると、重なった部分の色が消されます。

ｍｏｄｅの後ろのパレットコードを省略すると、ＣＯＬＯＲやＰＡＬＥＴステートメントで指定している色で表示されます。

パレットコードｃに２色合成の中間色を指定することができます。中間色は、２けたの１６進数で各けたが０～７のカラーコードで表わされます。

ＢＦは、ボックスフィルの指定で、長方形を描いて、中を塗りつぶすことを意味しています。タイリングパターンｔ＄は、中間色や色模様を出すための文字データです。詳しくは、ＰＡＩＮＴステートメントを参照してください。

座標の後ろに文字式を書くと、テキスト画面に対して文字で描くことができます。このとき描かれる長方形は、(x_1, y_1) と (x_2, y_2) の両座標間を対角とする文字の枠です。ただし、ＷＩＮＤＯＷステートメントによって座標系を変えている場合には注意が必要です。

〈注　意〉　パレットコードｃを数値変換で指定するときは、先頭がＢで始まる変数名を使うことができません。

参　照　ＬＩＮＥ

2.8

機能　折れ線を描きます。

書式

```
LINE〔(x1, y1)〕－(x2, y2)
      〔－(x3, y3)………－(xn, yn)〕
```

x_1, y_1：折れ線の始点座標。
x_2, y_2：折れ線の頂点座標。
x_3, y_3：折れ線の頂点座標。
………　　……………
xn, yn：折れ線の終点座標。
どれもWINDOWで指定したユーザー座標系の座標です。ただし、BASIC起動時の座標は次のように画面座標系と一致しています。

xi：横方向の座標。　WIDTH　40のとき、0～319。
WIDTH　80のとき、0～639。
yi：たて方向の座標。テキスト12行のとき、0～191。
または、0～383。
テキスト25行のとき、0～199。
(i＝1，2，3………，n)　または、0～399。

文例　LINE (100, 50) － (250, 50) － (200, 150) － (50, 150) － (100, 50)

⇨ (100, 50)、(250, 50)、(200, 150)、(50, 150) を実線で結んで、平行四辺形を描きます。

解説　このステートメントを実行すると、複数個の点を結んで、折れ線や多角形を描くことができます。

前のLINEステートメントの終点を始点とする場合は、(x_1, y_1) を省略することができます。

線の色はCOLORやPALETステートメントで指定している色となります。

描いた線だけを消すときは、黒色を指定して再度実行してください。

参照　LINE

2.8.6 POLY

機　　能　**正多角形を描きます。**

書　　式

```
POLY (x, y), r, c, △θ, θs, θe
```

x，y：正多角形の中心座標（WINDOWで指定したユーザー座標系）。
　BASIC起動時の座標は次のように画面座標系と一致しています。
　x：　横の座標。　WIDTH　40のとき、0～319の整数。
　　　　　　　　　WIDTH　80のとき、0～639の整数。
　y：　たての座標。テキスト12行のとき、　0～191の整数。
　　　　　　　　　　　　　　　　または、　0～383の整数。
　　　　　　　　　テキスト25行のとき、　0～199の整数。
　　　　　　　　　　　　　　　　または、　0～399の整数。

r：中心から頂点までの距離。
c：正多角形の辺の色。パレットコード。0～7の整数。
△θ：ステップ角（角度の増分）。省略すると1（度）。
θs：初期値。省略すると0（度）。 ┐
θe：終了角。省略すると360（度）。┘―図を参照。

省 略 形　POL.

文　　例　POLY (100, 100), 90, 4, 60, 0, 360
⇨（100，100）を中心とし、パレットコード4の色の正六角形を描きます。

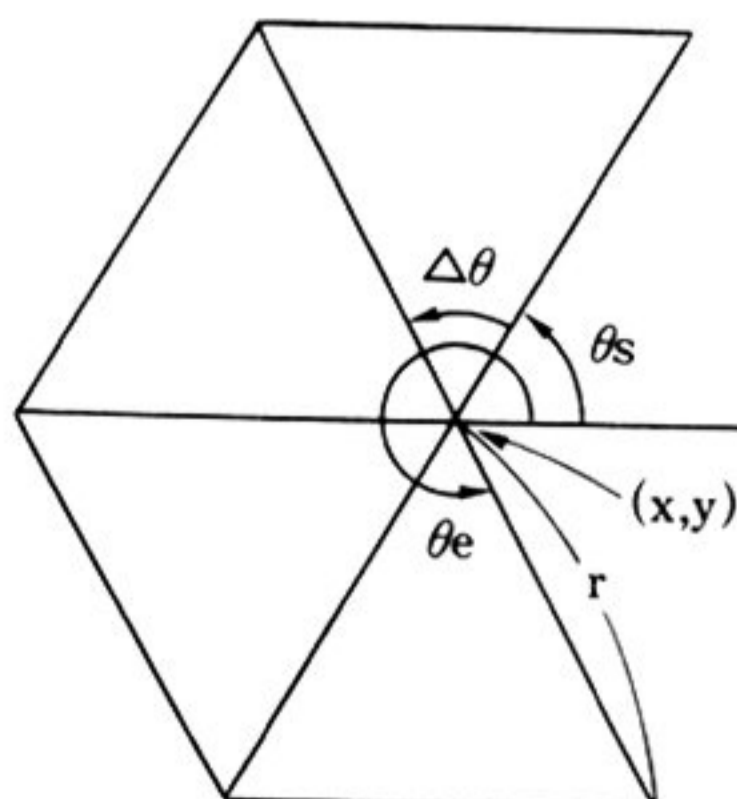
解　　説

POLYステートメントを実行すると、（x，y）を中心とし、中心から各頂点までの距離をrとする正多角形を描きます。

△θは、頂点間の角度でステップ角と呼び、120に指定すると正六角形、90に指定すると正方形となります。

一般的に正n角形を描くには

　$\triangle\theta = 360/n$　（度）
　$\theta e = \theta s + 360$

と指定します。

（x，y）：中心座標
r：中心から頂点までの距離
θs：初期角
θe：終了角
△θ：ステップ角

△θ＝0のときは、初期角θｓの頂点から円の中心（ｘ，ｙ）に長さｒの直線を引きます。また、△θ＝１４４、θｅ＝θｓ＋７２0とすると、☆形を描くことができます。

参照　2.8.7 ＣＩＲＣＬＥ

サンプルプログラム

（例１）３～１0角形の図形を様々な色で描きます。
スペースキーを押すと終了します。

```
10 ' POLY (例1)
20 INIT:WIDTH40,25,0:CLS4
30 FOR N=3 TO 10
40 C=INT(RND*7)+1
50 POLY(160,100),60,C,360/N,0,360
60 K$=INKEY$
70 IF K$=" " THEN INIT:CLS4:END
80 PAUSE10
90 CLS0
100 NEXT
110 GOTO 30
```

2.8

（例２）様々な色と大きさの星を表示します。

```
10 ' POLY (例2)
20 INIT:WIDTH40,25,0:CLS4
30 X=INT(RND*320)
40 Y=INT(RND*200)
50 R=INT(RND*50)
60 C=INT(RND*7)+1
70 POLY(X,Y),R,C,144,90,810
80 K$=INKEY$
90 IF K$=" " THEN INIT:CLS4:END
100 GOTO30
```

2.8.7 CIRCLE

機　　能　**円、楕円、および弧を描きます。**

書　　式

```
CIRCLE (x, y), r, c, f, θs, θe
```

x，y：円の中心座標（WINDOWで指定したユーザー座標系)。BASIC起動時の座標は次のように画面座標系と一致しています。

x：横の座標。　WIDTH　40のとき、0～319の整数。
　　　　　　　　WIDTH　80のとき、0～639の整数。

y：たての座標。テキスト12行のとき、　0～191の整数。
　　　　　　　　　　　　　　または、　0～383の整数。
　　　　　　　テキスト25行のとき、　0～199の整数。
　　　　　　　　　　　　　　または、　0～399の整数。

r：円の半径。
c：弧の色。パレットコード。0～7の整数。
f：偏平率。(たて径)／(横径) の値。
θs：初期角。省略すると0 (度)。
θe：終了角。省略すると360 (度)。

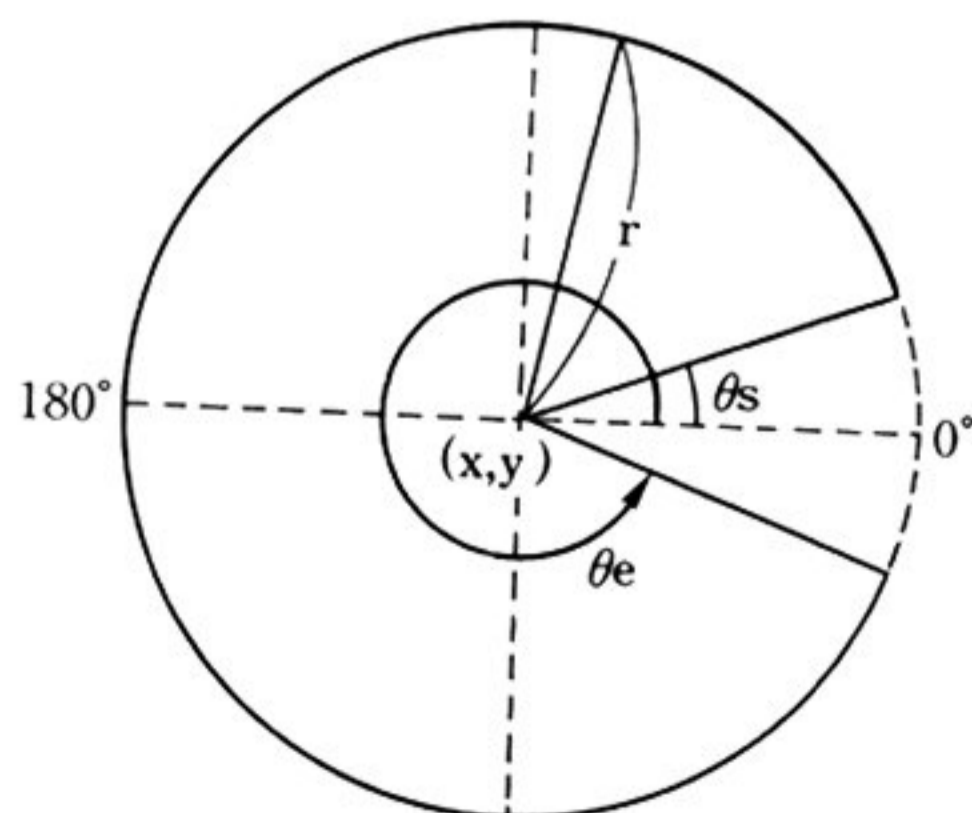
省 略 形　CI.

文　　例　CIRCLE (100, 100), 90, 4, 1, 0, 360

⇨ (100, 100) を中心とし、半径90、パレットコード4の色の円を描きます。

解　　説

CIRCLEステートメントを実行すると、(x, y) を中心とし、半径がrの円を描きます。

偏平率fを指定すると、楕円を描くことができます。fは、横径に対するたて径の比率でf＝1のとき、正円を描くことができます。また、偏平率f以降を省略しても、正円が描けます。

(x, y)：円の中心座標
r：円の半径
θs：初期角
θe：終了角

初期角θsと終了角θeを指定すると、その内角の弧を描くことができます。

〈注　　意〉 グラフィック画面が、解像度によって、半径ｒがたて方向、または横方向で倍として使われます。３２０×２００ドット、３２０×１９２ドットが標準で、３２０×３８４ドットがたて方向２倍、６４０×２００ドット、６４０×１９２ドットが横方向２倍、６４０×４００ドット、６４０×３８４ドットがたて横方向２倍となっています。
また、ＣＩＲＣＬＥステートメントで配列を使用した場合は、開始角、終了角には配列変数を使わないでください。

参　　照　2.8.6 ＰＯＬＹ

サンプルプログラム

色々な座標に色々な色の円または楕円を描くプログラムです。

```
10 ' CIRCLE (例)
20 INIT:WIDTH40,25,0:CLS4
30 X=INT(RND*320)
40 Y=INT(RND*200)
50 R=INT(RND*50)+1
60 C=INT(RND*7)+1
70 F=RND+0.5
80 CIRCLE(X,Y),R,C,F,90,810
90 K$=INKEY$
100 IF K$=" " THEN INIT:CLS4:END
110 GOTO30
```

扇形を描くには下記７２、７５番を追加し８０番を変更してください。

```
72  OS=INT (RND*360)
75  OE=INT (RND*360)
80  CIRCLE@(X, Y), R, C, F, -OS, -OE
```

2.8

2.8.8 CIRCLE@

機　　能　**円、楕円、扇形および弧を描きます。**

書　　式

```
CIRCLE@(x, y), r, c, f, θs, θe
```

x，y：円の中心座標（WINDOWで指定したユーザー座標）。BASIC起動時の座標は次のように画面座標系と一致しています。

x：横の座標。　WIDTH　40のとき、0～319の整数。
　　　　　　　 WIDTH　80のとき、0～639の整数。

y：たての座標。テキスト12行のとき、　0～191の整数。
　　　　　　　　　　　　　　または、　0～383の整数。
　　　　　　　テキスト25行のとき、　0～199の整数。
　　　　　　　　　　　　　　または、　0～399の整数。

r：円の半径。

c：弧の色。パレットコード0～7の整数。

f：偏平率。

θs：初期角。省略すると0（度）。

θe：終了角。省略すると360（度）。

省 略 形　CI.@

文　　例　CIRCLE@(100, 100), 90, 4

⇨（100，100）を中心とし、半径90、パレットコード4色の円を描きます。

解　　説　CIRCLEステートメントを実行すると、（x，y）を中心とし、半径がrの円を描きます。

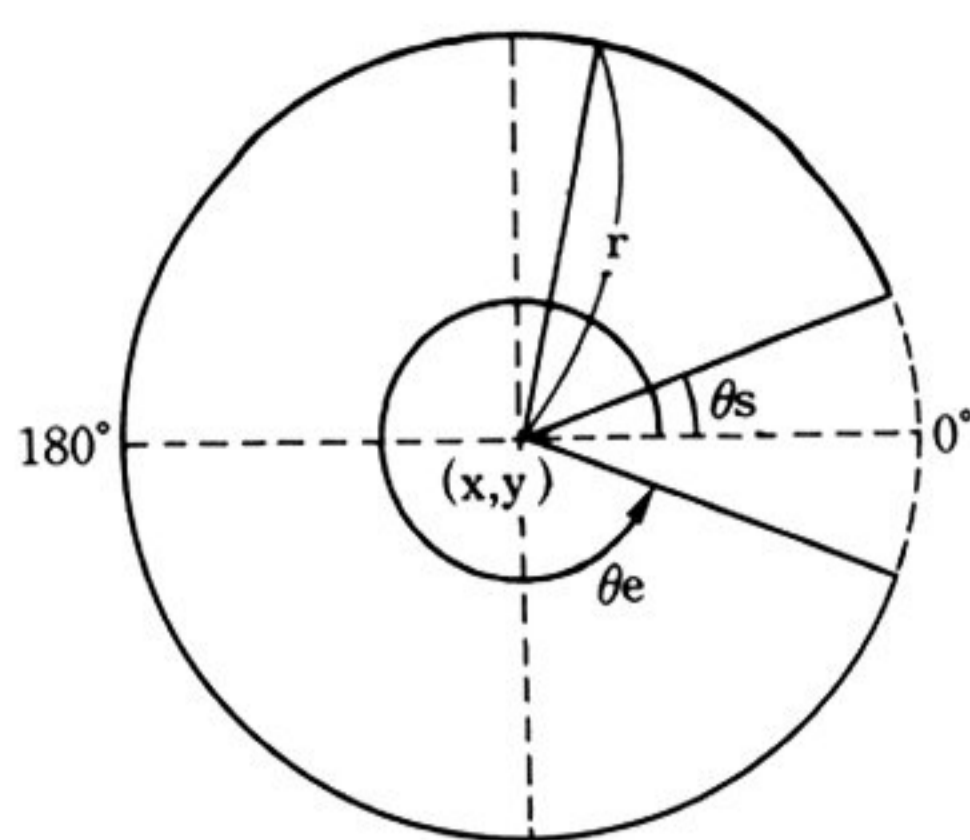
ただし、偏平率f＝1で、正しい円が描けるのは、グラフィック画面が320×200ドット、320×192ドット、640×400ドット、640×384ドットの場合で、その他の画面では楕円になります。

しかし、これらは、座標的には、グラフィック画面の座標と一致しています。

また、正しい円を描くには、

640×200ドット、640×192ドットでは偏平率fを0．5

320×400ドット、320×384ドットでは偏平率fを2に指定します。

偏平率が、1以下の場合には、半径rは、横方向の半径を意味し、1以上の場合には、半径rは、たて方向の半径を意味します。

(x, y) ：円の中心座標
r ：円の半径
θ s ：初期角
θ e ：終了角

初期値 θ sと終了角 θ eを指定すると、扇形または、弧を描くことができます。ただし、扇形は、θ sと θ eに負の値を指定する事により、弧は、正の値を指定する事で描けます。

参　照　2.8.7 CIRCLE

サンプルプログラム

(例1)

```
10 ' CIRCLE@ (例1)
20 INIT:WIDTH40,25,0:CLS4
30 X=INT(RND*320)
40 Y=INT(RND*200)
50 R=INT(RND*50)+10
60 C=INT(RND*7)+1
70 F=RND+.5
80 Os=INT(RND*360)
90 Oe=INT(RND*360)
100 CIRCLE@(X,Y),R,C,F,-Os,-Oe
110 A$=INKEY$
120 IF A$=" " THEN END
130 GOTO30
```

ランダムな扇形を描くプログラムです。

(例2)

```
10 ' CIRCLE@ (例2)
20 INIT:WIDTH40,25,0:CLS4
30 X=INT(RND*320)
40 Y=INT(RND*200)
50 R=INT(RND*50)+10
60 C=INT(RND*7)+1
70 F=RND+.5
80 CIRCLE@(X,Y),R,C,F,0,360
90 A$=INKEY$
100 IF A$=" " THEN END
110 GOTO30
```

ランダムな円や楕円を描くプログラムです。

2.8.9 PAINT

機　　能　**グラフィック画面上の閉曲線で囲まれた領域を色で塗りつぶします。**

書　　式

```
〔1〕PAINT (x, y), c, b〔, b1……, b6〕
〔2〕PAINT (x, y), t$, b〔, b1……, b6〕
```

x，y：塗りつぶしたい領域内の１点の座標（ＷＩＮＤＯＷで指定したユーザー座標系)。ＢＡＳＩＣ起動時の座標は次のように画面座標系と一致しています。

x：横の座標。　ＷＩＤＴＨ　４０のとき、０～３１９の整数。
　　　　　　　ＷＩＤＴＨ　８０のとき、０～６３９の整数。
y：たての座標。テキスト１２行のとき、　０～１９１の整数。
　　　　　　　　　　　　　または、　０～３８３の整数。
　　　　　　　テキスト２５行のとき、　０～１９９の整数。
　　　　　　　　　　　　　または、　０～３９９の整数。

c：塗る色。パレットコード（０～７の整数)、または中間色を表わす２けたの１６進数（＆Ｈ００～＆Ｈ７Ｆ)、および０～１２７の整数。
t＄：タイリングパターン。説明参照。
b，b_1～b_6：境界の色。パレットコード。０～７の整数。

省 略 形　ＰＡＩ.

文　　例　ＰＡＩＮＴ（０，０)，＆Ｈ４５，７
⇨（０，０）から境界のパレットコード７で囲まれた領域を緑とシアンの中間色で塗ります。

解　　説

ＰＡＩＮＴは、グラフィック画面上の閉曲線で囲まれた領域を指定された色あるいはタイリングパターンで塗りつぶすステートメントです。

（x，y）は、閉曲線内のドットの座標で、塗り始めの点を表わしています。

閉曲線はパレットコードｂで指定された色でなければなりません。

パレットコードｃに２色合成の中間色を指定することができます。中間色は、２けたの１６進数、各けたが０～７のカラーコードを表わしています。

（例）＆Ｈ４５………緑とシアン色のドットが交互にセットされた中間色を指定します。

タイリングパターンｔ＄は、中間色や色模様を出すための文字列データです。

最小のタイリングパターンは、たて×横が１×８で、これを指定するには、３バイトの文字列が必要です。なぜならば、１バイトのキャラクタコードのビットパターン８けたが、ちょうどたて×横＝１×８のドットパターンを表わし、これがカラー写真の３原色合成の原理と同様、Ｇ１（青の画面)、Ｇ２（赤の画面)、Ｇ３（緑の画面）の３枚分必要なので、結局、３バイトの文字列が必要となるのです。したがって、たて×横＝ｎ×８ドットのタイリングパターンを作

りたいときは、3×nバイトの文字列が必要で、最初の3バイトが1段目、次のバイトが2段目、………となります。ただし、n＝8までです。

文字列は、制御文字が表示できないこともあり、ＣＨＲ＄やＨＥＸＣＨＲ＄関数によって、用意するのが一般的です。

タイリングパターンｔ＄の例を次に示します。

（例1）ＨＥＸＣＨＲ＄（"ØØＦＦ55ØØＦＦＡＡ"）
………オレンジ色（黄と赤の中間色）を出します。

		青	赤	緑
奇数段：ØØＦＦ55	＝	00000000	11111111	01010101
偶数段：ØØＦＦＡＡ	＝	00000000	11111111	10101010

（例2）ＨＥＸＣＨＲ＄（"11ØØ1122ØØ2244ØØ4488ØØ88"）　………シアン色の斜線模様を出します。

		青	赤	緑
1段目：11ØØ11	＝	00010001	00000000	00010001
2段目：22ØØ22	＝	00100010	00000000	00100010
3段目：44ØØ44	＝	01000100	00000000	01000100
4段目：88ØØ88	＝	10001000	00000000	10001000

〈注　意〉

• 塗る色ｃを指定しない場合

ＣＯＬＯＲステートメントが実行されていれば、塗る色ｃは、文字表示色のカラーコードになります。塗る色ｃの指定または、ＣＯＬＯＲステートメントが実行されないと、その値が保持されます。

例）ＣＯＬＯＲ4　　⇨カラーコード4の色で画面がペイントされます。
ＰＡＩＮＴ（Ø，Ø）

• 境界色ｂを指定しない場合。（境界色はできるだけ指定してください）

イ）画面関係のステートメントで、カラーコードを指定するステートメント（ＬＩＮＥ、ＣＩＲＣＬＥ、ＣＯＬＯＲ、ＳＹＭＢＯＬ、ＣＩＲＣＬＥ@など）が実行されていた時は、最後に実行されたステートメントのカラーコードが境界色となります。

例）ＣＬＳØ

```
CIRCLE(2ØØ,1ØØ),5Ø,1
CIRCLE(2ØØ,1ØØ),8Ø,2
CIRCLE(2ØØ,1ØØ),9Ø,3
PAINT  (2ØØ,1ØØ),&H12
```

⇨境界色がカラーコード3として、ペイントされます。ですから一番大きな円の中をペイントします。

ここで、例にＣＯＬＯＲ1を（ＰＡＩＮＴステートメントの前に）入れると、一番小さい円だけペイントします。これは境界線がカラーコード1になったことを意味します。

ロ）画面全体が、カラーコードØ～7のどれか一色で塗られている時は、カラーコードØ～7で実行します。また、画面が中間色であった時は、中間色で使われているカラーコードを除いたカラーコードで実行します。

中間色の上に中間色をペイントする場合、実行しない時があります。

ハ）画面を一度ＣＬＳ０すれば、塗る色ｃまたは、タイリングパターンｔ＄で実行します。ただし、タイリングパターンｔ＄でｎ段目のＲ、Ｇ、Ｂがすべて０であった場合、タイリングペイントが途中で止まります。

2.8

2.8.10 PAINT@

機　能　**グラフィック画面上の任意の単色領域内を塗りつぶします。**

書　式

```
〔1〕PAINT@(x, y), c
〔2〕PAINT@(x, y), t$
```

x，y：塗りつぶしたい領域内の1点の座標（WINDOWで指定したユーザー座標系）。

BASIC起動時の座標は次のように画面座標系と一致しています。

x：横の座標。　WIDTH　40のとき、0～319の整数。
　　　　　　　WIDTH　80のとき、0～639の整数。

y：たての座標。テキスト12行のとき、　0～191の整数。
　　　　　　　　　　　　　または、　0～383の整数。
　　　　　　　テキスト25行のとき、　0～199の整数。
　　　　　　　　　　　　　または、　0～399の整数。

c：塗る色。パレットコード（0～7の整数）、または中間色を表わす2けたの16進数（&H00～&H7F）、および0～127の整数。

t$：タイリングパターン

2.8

省略形　PAI.@

文　例　PAINT@(0, 0), &H45

⇨座標（0, 0）から緑とシアンの中間色で塗りつぶします。

解　説

PAINT@は、グラフィック画面上の任意の単色で塗られた領域を、指定された色あるいはタイリングパターンで塗りつぶすステートメントです。

（x，y）は、閉曲線内のドットの座標で、塗り始めの点を表わしています。

パレットコードcに2色合成の中間色を指定することができます。中間色は、2けたの16進数で、各けた0～7のカラーコードを表わしています。

(例)　&H45………緑とシアン色のドットが交互にセットされた中間色を指定します。

また、座標（x，y）の色が塗りかえられる色で、その他の色が、境界色として使われます。したがって、（x，y）によって指定された場所がタイリングパターンだと1点だけしか塗りません。

2.8.11 POSITION

機　能　**PATTERN文でドットパターンを描くときの表示位置を指定します。**

書　式

```
POSITION x, y
```

x，y：ドットパターンの左上隅の座標（画面座標系）。
　x：横の座標。　WIDTH　40のとき、0～319の整数。
　　　　　　　　 WIDTH　80のとき、0～639の整数。
　y：たての座標。テキスト12行のとき、0～191の整数。
　　　　　　　　　　　　　　　　 または、0～383の整数。
　　　　　　　　 テキスト25行のとき、0～199の整数。
　　　　　　　　　　　　　　　　 または、0～399の整数。

省略形　POS.

文　例　POSITION　100，50

⇨PATTERN文によって設定されるドットパターンの表示開始座標を（100，50）とします。

解　説

POSITIONは、PATTERNステートメントとともに使い、PATTERNステートメントで描くドットパターンの左上隅の座標を指定します。

このステートメントは、必ずPATTERN文の前に書かれ、POSITIONとPATTERNの関係は、LOCATEとPRINTの関係によく似たものとなっています。

画面座標系の範囲外では表示しません。

参　照　2.8.12 PATTERN

2.8.12 PATTERN

機　　能　グラフィック画面にドットパターンを描きます。

書　　式

```
PATTERN n, x$〔, y$, ………〕
```

n：ドットパターンのたて方向の段数。1段1ドットで、－255～255の整数。
　nの値により積み重なる方向が変わります。
　n＜0のとき、下の方向に重なる段数。
　n＞0のとき、上の方向に重なる段数。
　n＝0のとき、「Illegl function call」のエラーが出ます。
x$, y$, ………：ドットパターンを表わす文字式。

省 略 形　PAT.

文　　例　PATTERN－8, HEXCHR$(A$)
⇨画面上にA$で表わされるドットパターンを表示します。

2.8

解　　説

PATTERNはPOSITIONを指定した座標を始点として、文字式で表わされたグラフィックのドットパターンを、たて方向にn段分表示します。

PATTERNとPOSITIONの関係は、PRINTとLOCATEの関係によく似ています。

文字式は文字列を表わし、x$, y$, ………というようにいくつかに分割して指定することができます。

1バイト（1文字）のキャラクタコードのビット・パターン8けたが、ちょうどたて×横＝1×8のドットパターンを表わします。

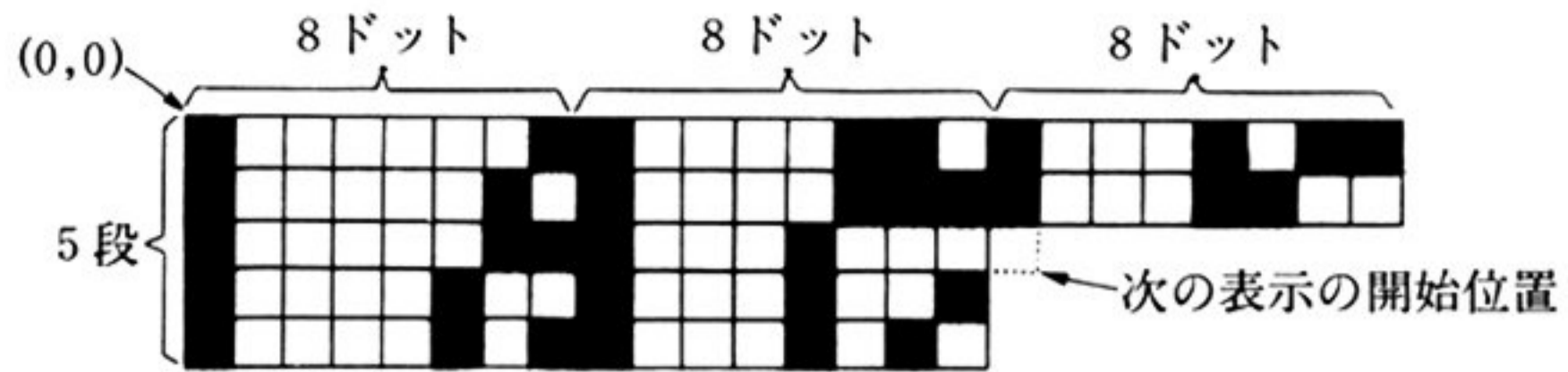
文字列は、制御文字が表示できないこともあり、CHR$やHEXCHR$関数によって用意するのが一般的です。

（例1）
```
100 POSITION 0, 0
110 PATTERN－1, CHR$(&HAA)
```

ドットパターン

AA ＝ 10101010 ⇨ ■□■□■□■□　　■がセット

ドットパターンの積み重ねは、nの値の符号によってその方向が決められ、nが負のとき上から下へ、nが正のとき下から上へ積み重ねられます。文字列が積み重ねの段数より多いときは、n段積み重ねた後、すぐ右隣りに移り、すべての文字列のドットパターンを表示してしまうまで同じ動作を繰り返します。

（例2）
```
100 A$=HEXCHR$("8182838485868788
              898A8B8C")
110 POSITION 0, 0
120 PATTERN－5, A$
```

ＫＡＮＪＩ＄関数を使って漢字を表示するときにＰＡＴＴＥＲＮを使用します。

（例3）１００　ＩＮＩＴ

１１０　ＰＯＳＩＴＩＯＮ１００，１００

１２０　ＰＡＴＴＥＲＮ－１６，ＫＡＮＪＩ＄（１６０６）

区点コード１６０６の漢字「愛」を本体の漢字ＲＯＭから引き出して表示します。

ＣＧＰＡＴ＄関数にシフトＪＩＳコードを指定して漢字や外字を表示するときにＰＡＴＴＥＲＮを使用します。

（例4）１００　ＩＮＩＴ

１１０　ＰＯＳＩＴＩＯＮ１００，１００

１２０　ＰＡＴＴＥＲＮ－１６，ＣＧＰＡＴ＄（＆Ｊ３０２６）

ＪＩＳ１６進コード３０２６の漢字「愛」を漢字ＲＯＭから引き出して表示します。

参　照　2.8.11 ＰＯＳＩＴＩＯＮ

2.8.13 SYMBOL

機　能　**画面上に文字列を指定の角度とサイズで表示します。**

書　式

```
〔1〕SYMBOL (x, y), x$, h, v, c, θ, mode
〔2〕SYMBOL (x, y), x$, h, v, t$, θ, mode
```

x，y：表示を始める座標（画面座標系）

　x：横の座標。ＷＩＤＴＨ　４０のとき、０～３１９の整数。
　　　　　　　ＷＩＤＴＨ　８０のとき、０～６３９の整数。

　y：たての座標。テキスト１２行のとき、０～１９１の整数。
　　　　　　　　　　　　　　　または　０～３８３の整数。
　　　　　　　　テキスト２５行のとき、０～１９９の整数。
　　　　　　　　　　　　　　　または　０～３９９の整数。

　※ｍｏｄｅに文字式か" "が指定されるとテキストの画面の座標とみなされます。

　　x：ＷＩＤＴＨ　４０のとき、０～３９の整数。
　　　ＷＩＤＴＨ　８０のとき、０～７９の整数。

　　y：テキスト１０行のとき、０～９の整数。
　　　テキスト１２行のとき、０～１１の整数。
　　　テキスト２０行のとき、０～１９の整数。
　　　テキスト２５行のとき、０～２４の整数。

x＄：表示する文字列。文字式。"文字列"。文字変数。

h：横方向の倍率。

v：たて方向の倍率。

c：文字の色。パレットコード。０～７の整数、または、中間色コード。
　＆Ｈ００～＆Ｈ７Ｆ。

t＄：タイリング・パターン。

θ：方向の指定。０～３の整数。

mode：ＰＳＥＴ、ＰＲＥＳＥＴ、ＸＯＲ、文字式、" "（ヌルストリング）
　　の４つがあります。

省 略 形　ＳＹ.

文　例　ＳＹＭＢＯＬ（１００，５０），"ＳＵＮ"，２，３，５，２，ＰＳＥＴ

⇨ＳＵＮという文字を画面上に横２倍・たて３倍のサイズ、パレットコード５の色、１８０度左回転させて表示します。

ＳＹＭＢＯＬ（０，０），"Ａ"，１，１，２，０，"Ａ"

⇨テキスト画面にＡという文字をＡというパターンで、たて横１倍のサイズ、標準方向で表示します。
テキスト表示の時はパレットコードは無視され、その時のテキストのカラーになります。

解　　説

ＳＹＭＢＯＬは、グラフィック画面上の任意の位置に文字を指定の方向・サイズ・色・モードで表示するステートメントです。

（ｘ，ｙ）は、文字列の表示を開始する左上のドットの座標を示します。

ｘ＄で表わされる文字列は、表示される文字列を示します。

横方向の倍率ｈおよびたて方向の倍率ｖは、表示する文字の横とたての倍率を指定します。

パレットコードｃは、表示する文字列の色を表わします。１６進数２けたの値を指定することによって、中間色を指定することができます。

(例)　＆Ｈ４５………緑とシアンの中間色を指定します。

θは文字の表示方向を示し、その値によって次のような指定ができます。

θの値	文字の表示方向
0	左から右（標準）
1	下から上（９０動左回転）
2	右から左（１８０度左回転）
3	上から下（２７０度左回転）

ｍｏｄｅは、文字列の表示モードを表わし、次の５つの設定ができます。

mode	意　　味
ＰＳＥＴ	文字列をそのまま表示する。
ＰＲＥＳＥＴ	文字列のパターンで消す。
ＸＯＲ	文字列と画面のドットをＸＯＲした結果を表示する。
ｙ＄	テキスト座標系のとき、表示される文字列を構成する１ドットは１キャラクタのサイズに拡大される。このとき、１キャラクタのパターンは、ｙ＄で表わされる文字列の先頭の１文字の図形文字となる。
" "	テキスト座標系のときこのステートメントに先立ってテキストの属性（色、反転、明滅など）やＫＳＥＮ（アンダーライン）が指定されていれば、その属性のついた文字列が表示される。

〈注　　意〉

このステートメントは、ＫＭＯＤＥ　０のときは、８×８の半角文字を、また、ＫＭＯＤＥ　１のときは、１６×８と１６×１６のドットパターンの文字を表示します。そのため、ＫＭＯＤＥ　１の場合、セミグラフィック文字は表示できません。(2.7.3 ＫＭＯＤＥ参照)。また、ＣＨＲ＄（０）も表示できません。表示にはグラフィック画面ではＷＩＮＤＯＷステートメントと座標、テキスト画面ではＣＯＮＳＯＬＥステートメントと座標に注意してください。

サンプル
プログラム

様々なCOMPUTERという文字を表示します。

```
10 ' SYMBOL (例)
20 INIT:WIDTH 40,25,0:KMODE 0:PRW &H2
30 SYMBOL(2,82),"COMPUTER",8,16,&H15,0,P
SET
40 CFLASH 1
50 SYMBOL(0,10),"COMPUTER",1,2,7,0,""
60 CFLASH
70 PRW &H22
80 FOR I=0 TO 3
90 SYMBOL(40,40),"COMPUTER",1,4,I+2,I,PS
ET
100 NEXT
110 KMODE 1
120 FOR I=0 TO 3
130 SYMBOL(200,40),"COMPUTER",1,2,I+2,I,
PSET
140 NEXT
150 PAUSE 10
160 PRW &H0
```

2.8.14 POINT

機　　能　**画面上の指定された点（ドット）のパレットコードを返します。**

書　　式

```
POINT (x, y)
```

ｘ，ｙ：画面上の座標（ＷＩＮＤＯＷ）で指定したユーザー座標系。ＢＡＳＩＣ起動時の座標は次のように画面座標系と一致しています。

ｘ：横方向の位置。ＷＩＤＴＨ　８０のとき、０～６３９の整数。
ＷＩＤＴＨ　４０のとき、０～３１９の整数。

ｙ：たての座標。　テキスト１２行のとき、０～１９１の整数。
または、０～３８３の整数。
テキスト２５行のとき、０～１９９の整数。
または、０～３９９の整数。

省 略 形　ＰＯＩ．

文　　例　ＩＦ ＰＯＩＮＴ（ｘ，ｙ）＝２　ＴＨＥＮ　１０００

⇨グラフィック画面上の点の座標（ｘ，ｙ）がパレットコード２であれば行番号１０００へジャンプします。

解　　説

画面（グラフィック）上の指定された座標のパレットコードが、この関数の値になります。パレットコードは０～７の整数です。（詳しくは、2.6.5 ＰＡＬＥＴステートメントを参照してください。）

（ｘ，ｙ）の点がＷＩＮＤＯＷステートメントで指定した領域の範囲外にあるときは、－１の値が返ります。

参　　照　3.3.2 ＳＣＲＮ＄

サンプルプログラム

ドットを正六角形の中でランダムウォークさせ、辺にぶつかったら正六角形の中心に戻るようにするプログラムです。

```
10 ' POINT (例)
20 INIT:WIDTH 40,25,0
30 POLY(160,100),30,1,60,0,360
40 PAINT(160,100),2
50 X=160:Y=100:I=160:J=100
60 R=INT(RND*4)+1
70 IF R=1 THEN X=X+1
80 IF R=2 THEN X=X-1
90 IF R=3 THEN Y=Y-1
100 IF R=4 THEN Y=Y+1
110 PSET(I,J,0)
120 IF POINT(X,Y)=1 THEN 50
130 PSET(X,Y,7)
140 I=X:J=Y
150 GOTO60
```

２０：画面の初期設定を行ないます。
３０：青い正六角形を描きます。
４０：正六角形内を赤で塗りつぶします。
５０：ドット座標の初期値を与えます。
６０～１５０：乱数の値によってドットを上下左右に動かします。
１２０：ドットが正六角形の辺の上に来たら６０番にジャンプします。

2.9 プリンタ制御ステートメント

2.9.1 CONSOLE#

機　能　プリンタ上での表示エリアを設定します。

書　式

```
CONSOLE#  Ys, Yl, [ , Xs, Xl]
```

Ｙｓ：たて方向の表示開始行。0～６５の整数。
Ｙｌ：たて方向の表示行数。１～６５の整数（正確には１～６５－Ｙｓ）。
Ｘｓ：横方向の表示開始けた。0～１１９の整数。
Ｘｌ：横方向の表示けた数。１～１９９の整数（正確には１～１１９－Ｙｓ）。

省略形　ＣＯＮＳ．＃

解　説

プリンタ用紙に対して、文字の印字エリアを設定します。
このステートメントの実行後は、指定した長方形のエリア内だけにＬＰＲＩＮＴ等で印字することができます。
設定後プリンタのヘッドは、（Ｘｓ，Ｙｓ）の位置へ移動します。
ＣＯＮＳＯＬＥ＃で表示エリアを設定しているとき、ＫＭＯＤＥを変更するのはやめてください。プリンタの横表示が正しく行なわれなくなります。
Ｙｌに0以外の値を設定すると、１度改ページを行ないます。
上の場合を除き各値（Ｙｓ，Ｙｌ，Ｘｓ，Ｘｌ）を設定すると１行改行します。
ＣＯＮＳＯＬＥ＃ステートメントで設定した値を解除するときは、ＣＯＮＳＯＬＥ＃またはＣＯＮＳＯＬＥ＃0，0，0，0としてください
ＣＯＮＳＯＬＥ＃は、そのまま、ＣＯＮＳＯＬＥ＃0，0，0，0は、１行改行します。

〈注　意〉　Ｘｓが0以外に設定され、ＫＭＯＤＥ0（純正プリンタでは、コード印字となる）のとき、ＨＣＯＰＹが最初の行だけ正しく印字されません。

参　照　2.3.9 ＬＰＲＩＮＴ

2.9

サンプル
プログラム

プリンタ上の横方向の表示けた数を３４に設定します。

```
10 ' CONSOLE# (例)
20 '
30 '  寒気ことのほか厳しい毎日が続いてお
ります。
40 '  しめやかな雨に心も落ち着く毎日とな
りました。
50 '  連日厳しい暑さが続いております。
60 '  日増しに秋も深まるようになりました
。
```

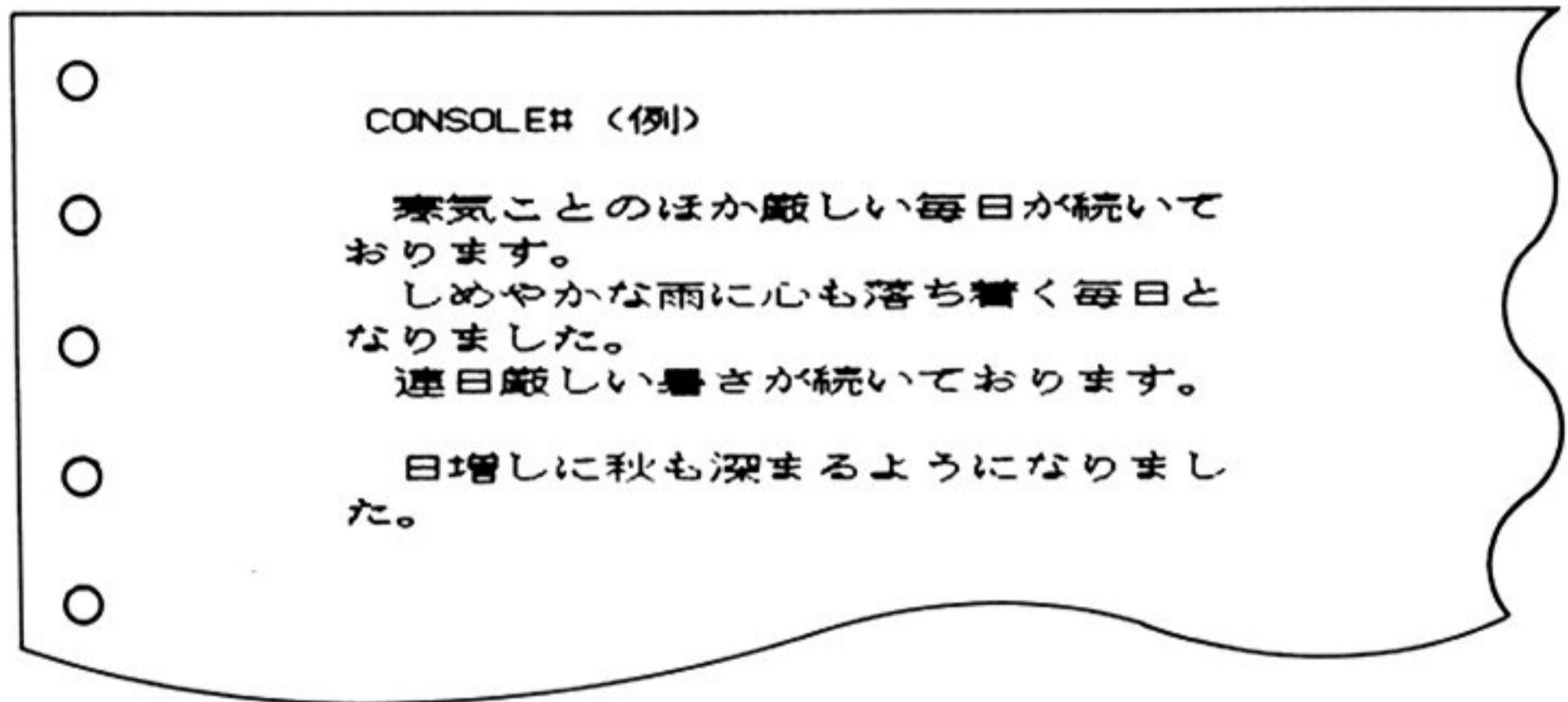
```
CONSOLE#,,0,34
Ok
```

```
LLIST*
```

```
         CONSOLE# (例)

          寒気ことのほか厳しい毎日が続いて
        おります。
          しめやかな雨に心も落ち着く毎日と
        なりました。
          連日厳しい暑さが続いております。

          日増しに秋も深まるようになりまし
        た。
```

2.9

2.9.2 LPOUT

機　　能　プリンタにコードを送ります。

書　　式

```
LPOUT x$
```

x＄：文字列

省 略 形　LPO.

文　　例　LPOUT　CHR＄（12）

⇨プリンタにコード12（ホームフィードのコード）を送ります。

解　　説　LPOUTは、プリンタに制御コードをそのまま送るためのステートメントです。プリンタの制御コードについては、各プリンタの説明書をご覧ください。

参　　照　2.3.9 LPRINT

2.9

2.9.3 HCOPY

機能

画面をプリンタにコピーします。

書式

HCOPY〔n〕

n：コピーする画面の指定。0～4の整数。

省略形

H.

解説

HCOPYは、画面をプリンタにハードコピーするためのステートメントです。

nの値によって、次のようにコピーする画面を選ぶことができます。

nの値	コピーできる画面
0	G1、G2、G3　（全グラフィック画面）
1	G1　（初期状態が青の画面）
2	G2　（初期状態が赤の画面）
3	G3　（初期状態が緑の画面）
4	全グラフィック画面とテキスト画面
省略	テキスト画面

複数画面の内の1枚を選んでコピーするには、まずSCREENステートメントで出力ページを指定してから行ないます。

〈注意〉

テキスト画面の行数指定が10または20の場合、HCOPYは無視されます。また、反転文字をハードコピーすることはできますが、拡大文字をハードコピーすることはできません。さらに、HCOPYは、通常CONSOLE#による設定を無視しますが、CONSOLE#ステートメントの注意に書かれた場合、正しく実行されなくなります。

参照

『ユーザーズマニュアル』の「日本語処理」2.9.1 CONSOLE#

サンプルプログラム

```
10 ' HCOPY (例)
20 INIT:WIDTH40,25
30 LOCATE0,1:PRINT"  日増しに秋も深まる
ようになりました。"
40 LINE(0,0)-(110,16),PSET,1,B
50 LINE(78,5)-(142,24),PSET,2,B
60 LINE(128,0)-(175,20),PSET,4,B
70 LOCATE0,10
```

２０：初期画面の設定。

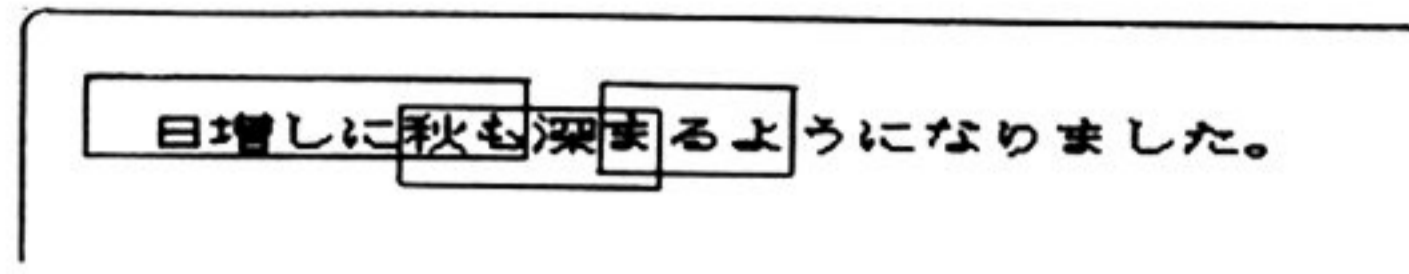
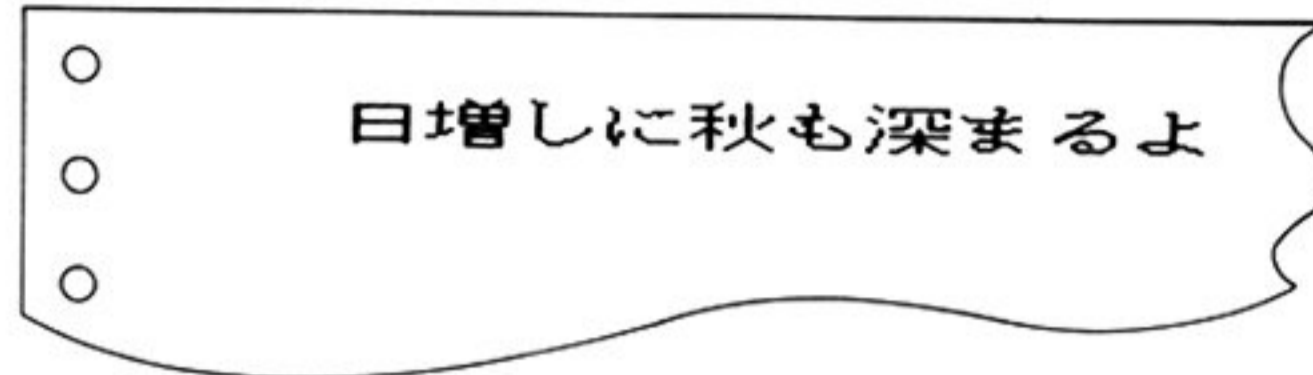
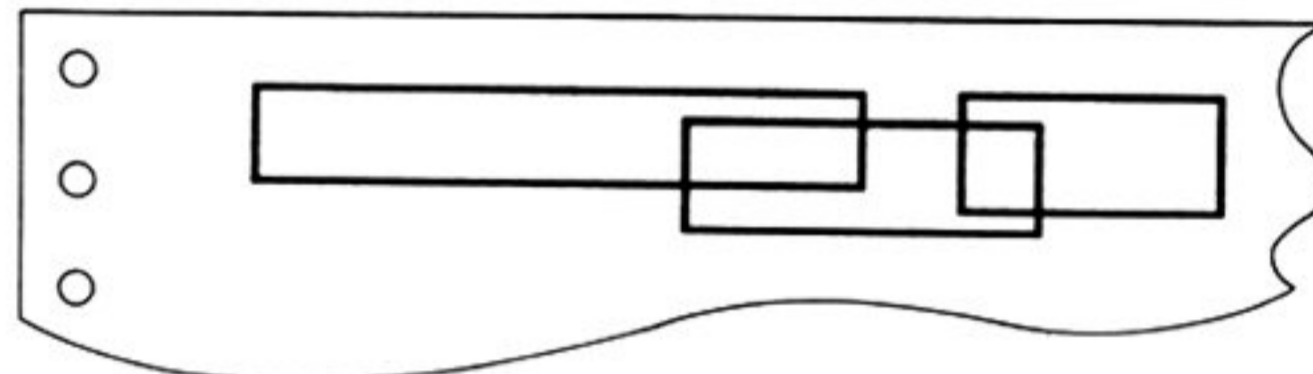
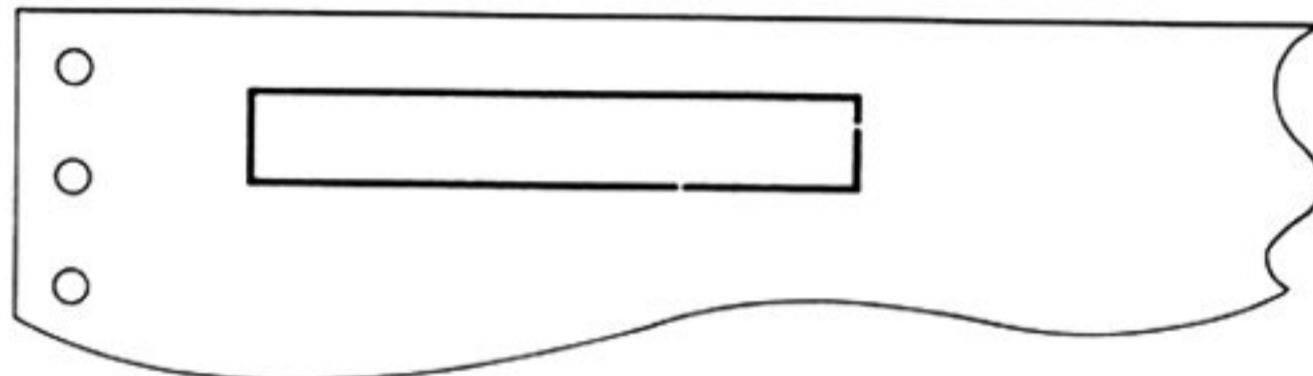
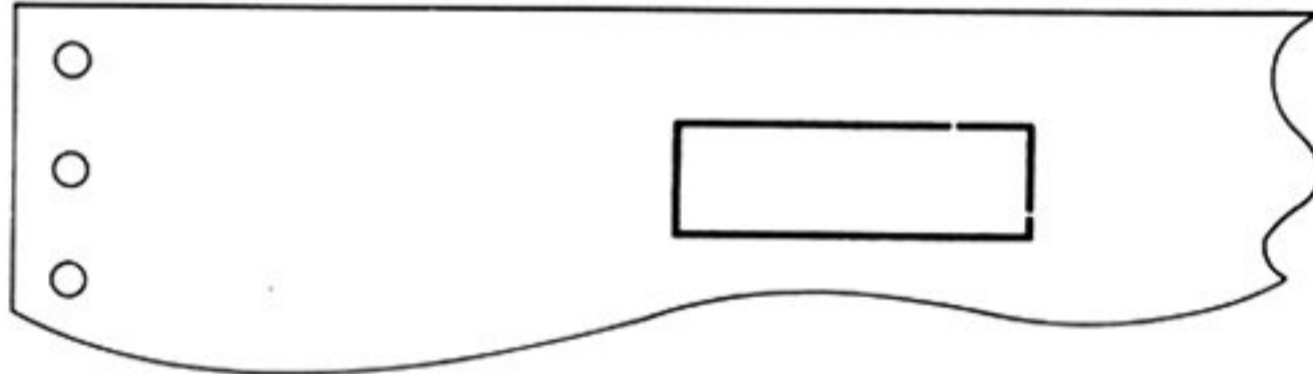
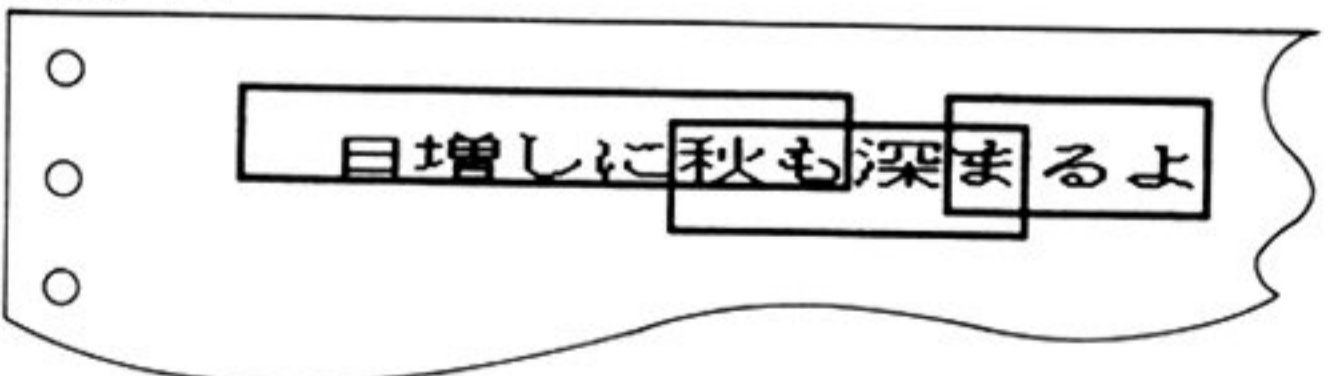
３０：テキスト画面に「日増しに秋も深まるようになりました。」と書きます。

４０：Ｇ１（青の画面）に長方形を描きます。

５０：Ｇ２（赤の画面）に長方形を描きます。

６０：Ｇ３（緑の画面）に長方形を描きます。

2.9

2.10 キー制御ステートメント

2.10.1 ＫＥＹ／ＤＥＦＫＥＹ

機　　能

ファンクションキーに文字列を定義します。

書　　式

〔1〕**ＫＥＹ　ｎ，文字式**
〔2〕**ＤＥＦＫＥＹ　ｎ，文字式**

ｎ：ファンクションキー番号。１～１０の整数。ファンクションキーとキー番号の対応は次の通りです。

文字式：最大１５文字までの文字列。文字列、ＣＨＲ＄関数を連結したものも可能。

省 略 形

［1］Ｋ.
［2］ＤＥＦＫ.

文　　例

ＫＥＹ２，"ＤＡＴＡ"
⇨ファンクションキーの２番目（Ｆ２）にＤＡＴＡという文字列を定義します。

解　　説

ＫＥＹおよびＤＥＦＫＥＹは、ｎ番のファンクションキーに文字列をセットするためのステートメントです。

このステートメントで文字列をセットすると、それ以後、セットされた文字列をファンクションキーのワンキー入力によって、キーボードから打ち込むことができるようになります。

コントロールコードは、ＣＨＲ＄関数を使って、文字列に連結するようにします。

ＢＡＳＩＣ起動時のファンクションキーの状態は次のようになっています。

番　号	定義されている文字列
1	"ＦＩＬＥＳ"＋ＣＨＲ＄（１３）
2	"？ＴＩＭＥ＄"＋ＣＨＲ＄（１３）
3	"ＫＥＹ　"
4	"ＬＩＳＴ"＋ＣＨＲ＄（２６，１３）
5	"ＲＵＮ　"＋ＣＨＲ＄（１３）
6	"ＬＯＡＤ　"＋ＣＨＲ＄（１３）
7	"ＷＩＤＴＨ　"
8	"ＣＨＲ＄（"
9	"ＰＡＬＥＴ　"
10	"ＣＯＮＴ"＋ＣＨＲ＄（１３）

※　キャラクタコードの１３は、キャリッジリターンコードで、ＣＨＲ＄（１３）は、「⏎キーを押す」と同じです。

ＣＨＲ＄（２６，１３）は「現在のカーソル以下の画面を消して、⏎キーを押す」と同じ動作をします。

参　照　2.10.2 ＫＬＩＳＴ

サンプルプログラム

```
KLIST

KEY 1,"FILES"+CHR$(13)
KEY 2,"?TIME$"+CHR$(13)
KEY 3,"KEY"
KEY 4,"LIST"+CHR$(26,13)
KEY 5,"RUN  "+CHR$(13)
KEY 6,"LOAD "+CHR$(13)
KEY 7,"WIDTH "
KEY 8,"CHR$("
KEY 9,"PALET "
KEY10,"CONT"+CHR$(13)
Ok
KEY 1,"DATA "
Ok
KEY 2,"LP.:LP."+CHR$(13)
Ok
KLIST

KEY 1,"DATA "
KEY 2,"LP.:LP."+CHR$(13)
KEY 3,"KEY"
KEY 4,"LIST"+CHR$(26,13)
KEY 5,"RUN  "+CHR$(13)
KEY 6,"LOAD "+CHR$(13)
KEY 7,"WIDTH "
KEY 8,"CHR$("
KEY 9,"PALET "
KEY10,"CONT"+CHR$(13)
Ok
```

2.10

2.10.2 KLIST／KEYLIST

機能　ファンクションキーの定義状態をリストします。

書式

〔1〕KLIST
〔2〕KEYLIST
〔3〕KLIST n
〔4〕KEYLIST n

n：0または1の整数。

省略形　［1］KL.
［2］K. L.

解説　ＫＬＩＳＴあるいはＫＥＹＬＩＳＴを実行すると、現在ファンクションキーに定義されている文字列を画面にリストします。
後ろに0か1の整数を指定すると、画面最下行にあるファンクションキーの機能表示を出したり、消したりすることができます。
（例1）ＫＬＩＳＴ　0……最下行の表示を消します。
（例2）ＫＬＩＳＴ　1……最下行の表示をつけます。
なお、ＢＡＳＩＣ起動時には、ＫＬＩＳＴ　1に初期設定されています。

参照　2.10.1 ＫＥＹ／ＤＥＦ　ＫＥＹ

サンプルプログラム

```
KLIST

KEY 1,"DATA "
KEY 2,"LP.:LP."+CHR$(13)
KEY 3,"KEY"
KEY 4,"LIST"+CHR$(26,13)
KEY 5,"RUN  "+CHR$(13)
KEY 6,"LOAD "+CHR$(13)
KEY 7,"WIDTH "
KEY 8,"CHR$("
KEY 9,"PALET "
KEY10,"CONT"+CHR$(13)
Ok
KEY 1,"FILES"+CHR$(13)
Ok
KEY 2,"?TIME$"+CHR$(13)
Ok
KLIST

KEY 1,"FILES"+CHR$(13)
KEY 2,"?TIME$"+CHR$(13)
KEY 3,"KEY"
KEY 4,"LIST"+CHR$(26,13)
KEY 5,"RUN  "+CHR$(13)
KEY 6,"LOAD "+CHR$(13)
KEY 7,"WIDTH "
KEY 8,"CHR$("
KEY 9,"PALET "
KEY10,"CONT"+CHR$(13)
Ok
```

2.10.11 ＫＥＹで再定義したＫＥＹ0，ＫＥＹ1を直接、初期設定にもどし、ＫＬＩＳＴにより表示させたものです。

2.10.3 ON KEY GOSUB

機　能　ファンクションキーからの割り込みが発生した場合、指定された行へ分岐します。

書　式

```
ON KEY GOSUB n1 [ , n2, n3, ……]
```

n_1，n_2，n_3，………………：ジャンプ先の番号。1～65534の整数。

省略形　O. K. GOS.

文　例　ON KEY GOSUB 100, 200, 300

⇨ファンクションキーの1番が押されたら行番号100へ、2番なら200へ、3番なら300へジャンプします。

解　説

「ON KEY GOSUB」は、i番のファンクションキーの割り込みが発生したとき行番号niにジャンプすることを定義するステートメントです。

すなわち、ファンクションキーの1が押されたとき、1番目の行番号n_1へ、
ファンクションキーの2が押されたとき、2番目の行番号n_2へ、
ファンクションキーの3が押されたとき、3番目の行番号n_3へ、
………………………………………………………………

というように、指定された分へ分岐します。

「ON KEY GOSUB」は、「ON 式 GOSUB」と本質的に異なった動作をするステートメントです。

というのは、「ON 式 GOSUB」を実行すると、式の値によってすぐに指定された行にジャンプしますが、「ON KEY GOSUB」を実行すると、ファンクションキーの割り込みが発生したときにどこへジャンプするのかを定義するだけで、すぐにジャンプするわけではありません。1度「ON KEY GOSUB」が実行されると、以降においてファンクションキーが押されたとき、プログラムの実行に割り込みがかかって指定された行へジャンプします。

ジャンプ先以降のサブルーチンからの復帰はRETURNによって行なわれます。単なるRETURNのときは、中断した所に戻って実行を再開し、RETURNの後ろに行番号を指定すると、その行から再開します。

なお、「ON KEY GOSUB」を実行すると、本来のファンクションキーの機能である文字列定義の機能が使用できなくなります。

ファンクションキーとその入力値の対応は次に示す通りです。

キー	値	キー	値
F1	1	SHIFT + F1	6
F2	2	SHIFT + F2	7
F3	3	SHIFT + F3	8
F4	4	SHIFT + F4	9
F5	5	SHIFT + F5	10

ファンクションキーの割り込みによってジャンプを行なうか否かは、KEY

2.10

ＯＮ／ＯＦＦ／ＳＴＯＰで制御することができます。（詳しくは、2.10.4参照）

参　照

2.10.4 ＫＥＹ　ＯＮ／ＯＦＦ／ＳＴＯＰ、2.2.21 ＯＮ　ＧＯＳＵＢ、
2.5.1 ＯＮ　ＥＲＲＯＲ　ＧＯＴＯ

サンプル
プログラム

次の例は、現在時刻を表示しながら、ファンクションキーの割り込みによって、内蔵の予約タイマーを起動するというプログラムで、予約タイマーから抜け出ると再び現在時刻の表示を始めます。

```
10 ' ON KEY GOSUB (例)
20 INIT:WIDTH40,25:CLS
30 ON KEY GOSUB90,90,90,90,90
40 LOCATE0,0
50 PRINT"年月日: ";DATE$
60 PRINT"曜日  : ";DAY$
70 PRINT"時刻  : ";TIME$
80 GOTO40
90 ASK
100 RETURN
```

２０：　初期画面の設定。

３０：　ファンクションキーのＦ１～Ｆ５のいずれかを押すと９０番にジャンプするよう定義します。

４０～８０　：　現在の年月日、曜日、時刻を表示します。

９０，１００：　内蔵の予約タイマーを起動するサブルーチンです。

2.10

2.10.4 KEY ON／KEY OFF／KEY STOP

機能 ファンクションキーによる割り込みの制御（許可、禁止、停止）をします。

書式

〔1〕KEY〔n〕 ON
〔2〕KEY〔n〕 OFF
〔3〕KEY〔n〕 STOP

n：ファンクションキーの番号。1～10の整数。

省略形

〔1〕K.〔n〕ON
〔2〕K.〔n〕OFF
〔3〕K.〔n〕S.

解説

KEY ON／OFF／STOPは、ファンクションキーが押されたときの割り込みを許可するか、禁止するか、停止するかを設定するステートメントです。

nには、1～10までのファンクションキーの番号を指定します。

「ON KEY GOSUB」ステートメントが定義されているとき、次の3つの設定が可能となります。

（1）KEYn ON このステートメントを実行後、ファンクションキーによる割り込みを許可します。
この状態のとき、ファンクションキーが押されるたびに割り込みが発生し、「ON KEY GOSUB」で定義されている行へジャンプします。

（2）KEYn OFF このステートメントを実行後、ファンクション・キーによる割り込みを禁止します。
この状態のとき、ファンクションキーを押しても、「ON KEY GOSUB」で定義された行へのジャンプは行ないません。

（3）KEYn STOP このステートメントを実行後、ファンクションキーによる割り込みを停止します。
この状態のとき、n番のファンクションキーを押しても、キーが押されたことを覚えているだけで、「ON KEY GOSUB」で定義された行へのジャンプは行ないません。しかし、その後「KEYn ON」が実施されると、割り込みが許可され先程のキーの入力に従って、定義された行へジャンプします。
nを省略すると、1～10のすべてのファンクションについて上記の機能をもちます。

参照 2.10.3 ON KEY GOSUB

2.10

サンプルプログラム

```
10 ' KEY ON/OFF/STOP (例)
20 INIT:WIDTH40,25:CLS
30 ON KEY GOSUB190,190,190,190,190
40 KEY1 OFF
50 KEY2 OFF
60 KEY3 OFF
70 KEY4 OFF
80 KEY5 ON
90 KEY6 OFF
100 KEY7 OFF
110 KEY8 OFF
120 KEY9 OFF
130 KEY10 OFF
140 LOCATE0,0
150 PRINT"年月日: ";DATE$
160 PRINT"曜日  : ";DAY$
170 PRINT"時刻  : ";TIME$
180 GOTO140
190 ASK
200 RETURN
```

２０：初期画面の設定

３０：ファンクションキーの F1 ～ F5 のいずれかを押すと１９０番にジャンプするように定義します。

４０～１３０：ファンクションキーの５番以外はすべて割り込みを禁止します。

１４０～１８０：現在の年月日、曜日、時刻を表示します。

１９０、２００：内蔵の予約タイマーを起動するサブルーチン。予約タイマーからの復帰は、 ESC キーを押してください。実行を終了するときは、 SHIFT ＋ BREAK キーを押してください。そのあと、ＫＥＹ　ＯＦＦを実行してください。 F5 がきかなくなっています。

2.10

2.10.5 CLICK ON/CLICK OFF

機能　キー入力時のクリック音の設定、解除を行ないます。

書式

〔1〕CLICK ON
〔2〕CLICK OFF

省略形

[1] CLI. ON
[2] CLI. OFF

解説

キーボードのキーを押すと、ボリュームが0でない限り、「カッ、カッ」という音が出ます。この音をクリック音といいます。

CLICK ON／OFFは、この音を出るようにしたり、止めたりするステートメントです。

(例1) CLICK OFF……クリック音が出なくなります。
(例2) CLICK ON………クリック音が出るようになります。

サンプルプログラム

次の例は、画面上のキャラクタを押したカーソルキーの方向に移動するプログラムです。

キー入力時のクリック音を止めるために30番でCLICK OFFを実行しています。

このプログラムはスペースキーを押すと終わることができます。

```
10 ' CLICK ON/OFF (例)
20 INIT:WIDTH 40,25,0
30 CLICK OFF
40 X=20:Y=12:I=20:J=12
50 K$=INKEY$(0)
60 IF K$=CHR$(28) THEN X=X+1
70 IF K$=CHR$(29) THEN X=X-1
80 IF K$=CHR$(30) THEN Y=Y-1
90 IF K$=CHR$(31) THEN Y=Y+1
100 IF K$=CHR$(32) THEN 190
110 IF X>38 THEN X=38
120 IF X<0  THEN X=0
130 IF Y>22 THEN Y=22
140 IF Y<0  THEN Y=0
150 LOCATE I,J:PRINT" "
160 LOCATE X,Y:PRINT"O"
170 I=X:J=Y
180 GOTO50
190 CLICK ON
200 END
```

30：　クリック音を止めます。
190：　クリック音が出るように設定します。

2.10.6 REPEAT ON／REPEAT OFF

機　　能　キー入力時のリピート機能の設定、解除を行ないます。

書　　式

〔1〕REPEAT　ON
〔2〕REPEAT　OFF

省 略 形

〔1〕REP.　ON
〔2〕REP.　OFF

解　　説

キーボードの同じキーを約1秒押していると、同じ文字を連続して入力することができるようになります。これをリピート機能といいます。

REPEAT　ON／OFFは、この機能の設定と解除を行なうステートメントです。

（例1）REPEAT　OFF……リピート機能が止まります。
（例2）REPEAT　ON………リピート機能が働くようになります。

サンプルプログラム

次はREPEAT　OFFの使用例で、INPUT文のデータ入力時に同じキーを押したままにしてもリピート機能が働きません。

```
10 ' REPEAT ON/OFF <例>
20 REPEAT OFF
30 INPUT"A=";A
40 INPUT"B=";B
50 PRINT"A+B=";A+B
60 PRINT"A-B=";A-B
70 PRINT"A*B=";A*B
80 PRINT"A/B=";A/B
90 REPEAT ON
100 END
```

20：　キー入力のリピート機能を止めます。
90：　リピート機能が働くよう設定します。

2.10

2.10.7 KEY∅

機　能　キーバッファに文字列を蓄えます。

書　式

```
KEY∅, x$
```

x＄：６３文字以内の文字列を表す文字式。単独の文字列、変数、配列、文字関数、およびそれらを連結したもの。

省略形　K. ∅

文　例　KEY∅, "MON"+CHR＄（１３）

⇨キーボードのキーを押さなくてもMON⏎が実行され、機械語モニタが起動します。

解　説

KEY∅は、キー・バッファ[1]に文字列を蓄えるためのステートメントです。

KEY∅を実行すると、ユーザーがキーボードのキーを押さなくても、あたかも押したようにx＄で指定した文字列が入力されます。

このステートメントを使うと、自己改変するプログラムを作ることができます。

x＄で指定できる文字列は最大６３文字までで、文字列が６４文字より長い時は、６４文字目以降が失われます。プログラム実行中は、KEY∅による入力文字はキーバッファに蓄えられているだけで、INPUTやINKEY＄などのキーの入力ステートメントを実行したときあるいは、プログラム実行を終わってコマンド待ちになったときに、はじめて受けられます。

１）キーバッファ……キーボードから打ち込んだ文字データを一時的にためておく記憶領域です。
キーボードからの入力は割り込み処理によって、このバッファに記憶されます。

参　照　2.10.1 KEY

2.10

サンプル
プログラム

次はキー入力した文字型データを自分のＤＡＴＡ文のデータとして持って自己増殖を行なうプログラムの例です。

```
10 ' KEY0 (例)
20 KMODE1
30 NM$=""
40 PRINT CHR$(30,30)
50 INPUT"名前 ";NM$
60 IF NM$="" THEN 150
70 RESTORE1000
80 DL=1000
90 READ A$
100 IF A$<>"*" THEN DL=DL+10:GOTO90
110 PRINT STR$(DL);" DATA ";NM$
120 PRINT STR$(DL+10);" DATA *"
130 KMODE0
140 KEY0,CHR$(30,30,30,13,13)+"GOTO20"+C
HR$(13):END
150 RESTORE1000
160 READ A$
170 IF A$="*" THEN PRINT CHR$(31):GOTO20

180 PRINT A$
190 GOTO160
1000 DATA *

RUN
名前 ? 田中雄一
 1000 DATA 田中雄一
 1010 DATA *
名前 ? 井上彩子
 1010 DATA 井上彩子
 1020 DATA *
名前 ? 今井友子
 1020 DATA 今井友子
 1030 DATA *
名前 ?
田中雄一
井上彩子
今井友子

名前 ?
Break in 50
Ok.
LIST
10 ' KEY0 (例)
20 KMODE1
30 NM$=""
40 PRINT CHR$(30,30)
50 INPUT"名前 ";NM$
60 IF NM$="" THEN 150
70 RESTORE1000
80 DL=1000
90 READ A$
100 IF A$<>"*" THEN DL=DL+10:GOTO90
110 PRINT STR$(DL);" DATA ";NM$
120 PRINT STR$(DL+10);" DATA *"
130 KMODE0
140 KEY0,CHR$(30,30,30,13,13)+"GOTO20"+C
HR$(13):END
150 RESTORE1000
160 READ A$
170 IF A$="*" THEN PRINT CHR$(31):GOTO20

180 PRINT A$
190 GOTO160
1000 DATA 田中雄一
1010 DATA 井上彩子
1020 DATA 今井友子
1030 DATA *
Ok
```

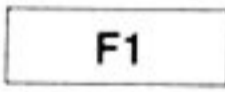
2.10.8 KBUF ON／KBUF OFF

機能　キーバッファの使用の許可・禁止の設定をします。

書式

```
〔1〕KBUF ON
〔2〕KBUF OFF
```

省略形　〔1〕KB. ON
〔2〕KB. OFF

文例　KBUF OFF
⇨プログラム実行中、入力ステートメントまでキー入力を禁止します。

解説　キーバッファの使用の許可・禁止の指定をするステートメントです。

〔1〕KBUF ON……キー入力のデータを蓄えることを許可することで、プログラム実行中でも、データやコマンドを蓄えることができます。

〔2〕KBUF OFF…キー入力のデータを入力ステートメントが来るまでキーバッファに蓄えることを禁止して、入力ステートメントになったときだけキー入力を受けつけます。

(例)
```
10 CLS
20 KBUF OFF
30 FOR I=0 TO 1000:NEXT
40 LINPUT A$
50 PRINT A$
60 END
```

この例では、RUN⏎後すぐにファンクションキー入力をしても40番が実行されるまでキー入力は有効になりません。

たとえば、RUN実行後すぐにファンクションキー F1 を入力しても何も表示せずに入力待ちになり、20番目をKBUF ONに変えると「FILES」を2回表示し、終了します。

BASC起動時は、KBUF ONになっています。また、KBUF OFFは、KEY0, " "と同じです。" "はヌルストリングを示しています。

参照　2.10.7 KEY0

2.10

2.11 通信回線制御ステートメント

2.11.1 ON COM GOSUB

機能　通信回線からの割り込みが発生したときに、指定の行へ分岐します。

書式

```
〔1〕ON COM GOSUB n
〔2〕ON COM GOSUB L$
```

n：行番号。ジャンプ先の行番号。

L$：ラベル名。"文字列"ジャンプ先のラベル名。LABELで定義されたもの。

文例　ON COM GOSUB 1000

⇨「COM ON」の状態のとき、RS－232Cのポートから割り込みが発生すると、行番号1000にジャンプします。

解説　「ON COM GOSUB」は、通信回線（RS－232Cのポート）の割り込みが発生したとき、指定した行にジャンプすることを定義するステートメントです。

ジャンプ先は、サブルーチンを形成し、何らかの処理を行なった後、RETURNによってメインルーチンに復帰します。

単にRETURNとしたときは、中断した時点から実行再開し、RETURNの後ろの行番号、ラベル名を指定するとその行から再開します。

通信回線の割り込みによってジャンプを行なうか否かは、COM ON／OFF／STOPで制御することができます。(詳しくは2.11.2参照)

COM ON ……通信回線の割り込みを許可してジャンプを行ないます。

COM OFF……通信回線の割り込みを禁止しています。

COM STOP…通信回線の割り込みを停止してジャンプを行ないません。

1度「ON COM GOSUB」を実行すると「COM ON」の状態になります。

参照　2.11.2 COM ON／OFF／STOP、2.10.3 ON KEY GOSUB、2.13.1 ON TIME$ GOSUB

サンプルプログラム　自分のコンピュータと相手のコンピュータをRS－232Cケーブルで接続して、それぞれでこのプログラムを走らせると、キーボードから打ち込むメッセージによる会話ができます。

```
10 ' ON COM GOSUB (例)
20 WIDTH40,25:CLS
30 Y=13:X=1
40 OPEN"C",#1,"COM:5N83X"
50 ON COM GOSUB "受信ルーチン"
60 LOCATE 0,12:PRINT"受信メッセージ"
70 LOCATE 0,1:PRINT"送信メッセージ"
80 COM STOP
90 K$=INKEY$:IF K$<>"" THEN PRINT#1,K$;:PRINT K$;
100 COM ON:GOTO80
110 LABEL"受信ルーチン"
120 X1=POS(0):Y1=CSRLIN
130 LOCATE X,Y
140 IF LOC(1)<>0 THEN PRINT INPUT$(LOC(1),#1);
150 X=POS(0):Y=CSRLIN
160 LOCATE X1,Y1
170 RETURN
```

接続ケーブルは『ユーザーズマニュアル』の「ＲＳ－２３２Ｃインターフェイスの使い方」の項を参照してください。

2.11

2.11.2 COM ON／COM OFF／COM STOP

機能

通信回線からの割り込みの制御（許可、禁止、停止）をします。

書式

```
〔1〕COM ON
〔2〕COM OFF
〔3〕COM STOP
```

文例

ＣＯＭ ＯＦＦ

⇨通信回線からの割り込みを禁止します。これを実行すると「ＣＯＭ ＯＮ」が実行されるまで割り込みが禁止されます。

解説

ＣＯＭ ＯＮ／ＯＦＦ／ＳＴＯＰは、通信回線（ＲＳ－２３２Ｃポート）に外部からの通信が入ったことによる割り込みを許可するか、禁止するか、停止するかを設定するステートメントです。

「ＯＮ ＣＯＭ ＧＯＳＵＢ」ステートメントが定義されているとき、次の３つの設定が可能となります。

（１）ＣＯＭ ＯＮ…このステートメントを実行後、通信回線による割り込みを許可します。この状態のとき、通信回線にデータが入るたびに割り込みが発生し、「ＯＮ ＣＯＭ ＧＯＳＵＢ」で定義されている行へジャンプします。

（２）ＣＯＭ ＯＦＦ…このステートメントを実行後、通信回線による割り込みを禁止します。この状態のとき、通信があっても、「ＯＮ ＣＯＭ ＧＯＳＵＢ」で定義された行へのジャンプは行ないません。

（３）ＣＯＭ ＳＴＯＰ このステートメントを実行後、通信回線による割り込みを停止します。
この状態のとき、通信回線にデータが入っても、データの入力を覚えているだけで、「ＯＮ ＣＯＭ ＧＯＳＵＢ」で定義された行へのジャンプは行ないません。しかしその後「ＣＯＭ ＯＮ」が実行されると、割り込みが許可され、先程のデータに従って、定義された行へジャンプします。

2.11

参照

2.11.1 ＯＮ ＣＯＭ ＧＯＳＵＢ

サンプルプログラム

ＯＮ ＣＯＭ ＧＯＳＵＢのサンプルプログラム参照。

2.12 マウス制御ステートメント

2.12.1 MOUSE

機能 マウスの諸機能を設定します。

書式

```
〔1〕MOUSE0
〔2〕MOUSE1〔,Sx〕〔,Sy〕
〔3〕MOUSE2,d,r
〔4〕MOUSE3,Sx1,Sy1,Sx2,Sy2
```

Ｓｘ，Ｓｙ：マウスカーソルの初期座標。

Ｓｘ：－３２７６８～６５５３５

Ｓｙ：－３２７６８～６５５３５

ｄ：マウスカーソルの移動比率の方向。０または１。

０：Ｘ方向の移動比率のセット。

１：Ｙ方向の移動比率のセット。

ｒ：マウスカーソルの移動比率。１～３２。

Sx_1，Sy_1：マウスカーソルの移動範囲の始点座標。

Sx_2，Sy_2：マウスカーソルの移動範囲の終点座標。

Sx_1、Sx_2：－３２７６８～６５５３５

Sy_1、Sy_2：－３２７６８～６５５３５

省略形 ＭＯＵ．

文例 ＭＯＵＳＥ１，０，０

⇨マウス機能をＯＮにし、マウスカーソルの初期座標を（０，０）に設定します。

ＭＯＵＳＥ２，１，１０

⇨たて方向の移動比率を１０とし、マウスから送られてくる値を１０で割り、その値をＭＯＵＳＥ（１）の値に加えます。

解説 マウスの諸機能を設定します。

次のような設定が可能です。

(1) ＭＯＵＳＥ０

マウス機能をＯＦＦ状態にします。マウスの使用後には、必ずこのステートメントを実行してください。

(2) ＭＯＵＳＥ１

マウス機能をＯＮ状態にし、初期座標を指定します。

Ｓｘで指定された値がＭＯＵＳＥ関数のＭＯＵＳＥ（０）に、Ｓｙで指定された値がＭＯＵＳＥ（１）にセットされます。

Ｓｘ、Ｓｙを省略すると、マウス機能をＯＮ状態にするだけとなります。

(3) ＭＯＵＳＥ２

マウスカーソルの移動比率を指定するステートメントです。

移動比率 r は、１～３２の値を指定します。

方向 d＝0 の場合は横方向、１の場合はたて方向の移動比率をセットできます。

(例) ＭＯＵＳＥ２，０，１２⇨横方向の移動比率を１２としてマウスから送られてくる値を本体内で１２で割り、それをＭＯＵＳＥ（０）の値に加えます。

(4) ＭＯＵＳＥ３

マウスカーソルの移動範囲を設定するステートメントです。

（Sx_1，Sy_1）と（Sx_2，Sy_2）を対角とする四角形で囲まれた領域の内部をカーソル移動の範囲とします。ただし、ＢＡＳＩＣ起動時は $Sx_1 = Sy_1 = 0$、$Sx_2 = 319$、$Sy_2 = 119$ となっています。

参　照

2.12.2 ＭＯＵＳＥ関数

サンプルプログラム

次のプログラムは、マウスを使って画面に線画を描くものです。

実行前にマウスを接続しなければ作動しません。

```
10 ' MOUSE (例)
100 INIT:WIDTH80,25,0,0
110 MOUSE0
120 MOUSE1,0,0
130 CLS0:X=0:Y=0:XX=0:YY=0
140 LINE(XX,YY)-(XX,YY),XOR,7
150 LOCATE0,0:PRINTUSING"X=#### Y=####",XX,YY;
160 X=MOUSE(0):Y=MOUSE(1)
170 LINE(XX,YY)-(XX,YY),XOR,7
180 IF MOUSE(2,1)=-1 THEN LINE(XX,YY)-(X,Y),PSET,7
190 XX=X:YY=Y
200 IF MOUSE(2,2)=-1 THEN CLS0:GOTO140
210 GOTO140
```

2.12

2.12.2 MOUSE（関数）

機能　マウスの諸状態を返します。

書式

```
MOUSE (n〔, b〕)
```

n：0～8の整数。
b：ボタン番号。1：左。
　　　　　　　2：右。

省略形　MOU.

文例　PRINT MOUSE (0), MOUSE (1)
⇨マウスカーソルの現在の座標（X, Y）を表示します。

解説　MOUSEは、マウスの各種の状態を返す関数です。
nとbの値によって、次のような設定ができます。

（1）MOUSE (0)
　マウスカーソルの現在のx座標（横方向の座標）を返します。
（2）MOUSE (1)
　マウスカーソルの現在のy座標（たて方向の座標）を返します。
（3）MOUSE (2, b)
　指定されたボタン番号bのボタンが押されているときに－1を返し、離されているとき0を返します。
（4）MOUSE (3, b)
　ボタン番号bのボタンが押された時点のマウスカーソルのx座標（横方向の座標）を返します。
（5）MOUSE (4, b)
　ボタン番号bのボタンが押された時点のマウスカーソルのy座標（たて方向の座標）を返します。
（6）MOUSE (5, b)
　ボタン番号bのボタンが離された時点のマウスカーソルのx座標（横方向の座標）を返します。
（7）MOUSE (6, b)
　ボタン番号bのボタンが離された時点のマウスカーソルのy座標（たて方向の座標）を返します。
（8）MOUSE (7)
　最後にこの関数が呼び出されてから、次にこの関数が呼び出されるまでにマウスカーソルが動いたx方向（横方向）の移動距離を返します。
（9）MOUSE (8)
　最後にこの関数が呼び出されてから、次にこの関数が呼び出されるまでにマウスカーソルが動いたy方向（たて方向）の移動距離を返します。

参照　2.12.1 MOUSE

2.13 タイマー制御ステートメント

2.13.1 ON TIME$ GOSUB

機能 **タイマーによる割り込みが発生したときに、指定の行へ分岐します。**

書式

〔1〕ON TIME$="hh：mm〔／ii〕"GOSUB n
〔2〕ON TIME$="hh：mm〔／ii〕"GOSUB L$

n：行番号。ジャンプ先の行番号。
L$：ラベル名。ジャンプ先のラベル名。LABELで定義されたもの。
"hh：mm〔／ii〕"：ジャンプを開始する時刻と次にジャンプするまでの間隔。
hh：時。00～23の2けたの整数。
mm：分。00～59の2けたの整数。
ii：分。00～60の2けたの整数で次にジャンプを行なうまでの分数を表わす。

文例 〔1〕ON TIME$="12：00" GOSUB 1000
⇨「TIME$ ON」の状態のとき、時刻がちょうど12時になって割り込みが発生すると、行番号1000にジャンプします。
〔2〕ON TIME$="07：00／02" GOSUB 1000
⇨「TIME$ ON」の状態のとき、時刻がちょうど7時になって割り込みが発生すると行番号1000にジャンプします。その後2分間隔で1000番にジャンプします。

解説 「ON TIME$ GOSUB」は、タイマーによる割り込みが発生したとき、指定した行にジャンプすることを定義するステートメントです。

ジャンプする時刻は"hh：mm"で何時何分まで設定できます。hh：mmの後ろに／iiを指定するとhh時mm分以後ii間隔でジャンプします。

ジャンプ先は、サブルーチンを形成し、何らかの処理を行なった後、RETURNによってメインルーチンに復帰します。

単にRETURNとしたときは、中断した時点から実行再開し、RETURNの後の行番号、ラベル名を指定するとその行から再開します。

タイマーの割り込みによってジャンプを行なうか否かは、TIME$ ON／OFF／STOPで制御することができます。(詳しくは2.13.2参謄照)

TIME$ ON ……タイマーの割り込みを許可してジャンプを行ないます。
TIME$ OFF……タイマーの割り込みを禁止します。
TIME$ STOP…タイマーの割り込みを停止してジャンプを行ないません。

「ＯＮ　ＴＩＭＥ＄　ＧＯＳＵＢ」を使用しないときは、

　　ＯＮ　ＴＩＭＥ＄＝""

を実行してください。本機のＴＩＭＥＲランプを消すことができます。

参　　照

2.13.2 ＴＩＭＥ＄　ＯＮ／ＯＦＦ／ＳＴＯＰ、
2.13.3 ＯＮ　ＫＥＹ　ＧＯＳＵＢ、
2.11.1 ＯＮ　ＣＯＭ　ＧＯＳＵＢ

2.13.2 ＴＩＭＥ＄ ON／ＴＩＭＥ＄ OFF／ＴＩＭＥ＄ STOP

機能　**タイマーによる割り込みの制御（許可、禁止、停止）をします。**

書式

```
〔1〕TIME$ ON
〔2〕TIME$ OFF
〔3〕TIME$ STOP
```

解説

ＴＩＭＥ＄　ＯＮ／ＯＦＦ／ＳＴＯＰは、タイマーからの割り込みを許可するか、禁止するか、停止するかを設定するステートメントです。

「ＯＮ　ＴＩＭＥ＄　ＧＯＳＵＢ」ステートメントが定義されているとき、次の3つの設定が可能となります。

（1）ＴＩＭＥ＄　ＯＮ…このステートメントを実行後、タイマーによる割り込みを許可します。この状態のとき、設定時刻になるたびに割り込みが発生し、「ＯＮ　ＴＩＭＥ＄　ＧＯＳＵＢ」で定義されている行へジャンプします。

（2）ＴＩＭＥ＄　ＯＦＦ…このステートメントを実行後、タイマーによる割り込みを禁止します。
この状態のとき、設定時刻になっても、「ＯＮ　ＴＩＭＥ＄　ＧＯＳＵＢ」で定義された行へのジャンプは行ないません。

（3）ＴＩＭＥ＄　ＳＴＯＰ…このステートメントを実行後、タイマーによる割り込みを停止します。
この状態のとき、設定時刻になっても、タイマーによる割り込みを覚えているだけで、「ＯＮ　ＴＩＭＥ＄　ＧＯＳＵＢ」で定義された行へのジャンプは行ないません。しかし、その後「ＴＩＭＥ＄　ＯＮ」が実行されると、割り込みが許可され、先程のデータに従って、定義された行へジャンプします。

参照　2.11.1 ＯＮ　ＣＯＭ　ＧＯＳＵＢ、2.10.3 ＯＮ　ＫＥＹ　ＧＯＳＵＢ

2.13

2.14.1 ＴＶＰＷ

機能

専用ディスプレイテレビの電源のｏｎ／ｏｆｆを行ないます。

書式

〔1〕**ＴＶＰＷ　ＯＮ**
〔2〕**ＴＶＰＷ　ＯＦＦ**

省略形

〔1〕ＴＶ．ＯＮ
〔2〕ＴＶ．ＯＦＦ

解説

専用ディスプレイテレビの電源を入れたり切ったりするステートメントです。
(例１) ＴＶＰＷ　ＯＮ⏎……専用ディスプレイテレビの電源が入ります。
(例２) ＴＶＰＷ　ＯＦＦ⏎……専用ディスプレイテレビの電源が切れます。
ＴＶＰＷ　ＯＮ⏎によって専用ディスプレイテレビの電源をＯＮにしたときは、テレビ放送が表示されるので、コンピュータのディスプレイ画面として使用するときは、SHIFTキーを押しながらテンキーの・を押すか、ＣＲＴ１⏎を実行してコンピュータ画面にする必要があります。テレビの電源スイッチを入れたり切ったりすることはやめてください。

参照

『ユーザーズマニュアル』の「テレビコントロール」、2.14.2 ＣＲＴ

2.14.2 CRT

機　能

テレビの映像画面とコンピュータ画面およびスーパーインポーズ画面の切り換えをします。

書　式

```
CRT n
```

n：0～3の整数。

文　例

CRT3
⇨テレビとコンピュータ画面を同じコントラストで重ねて表示します。

解　説

使用モニターが専用ディスプレイテレビのとき、このステートメントの実行によって、テレビ映像画面とコンピュータ画面の切り換えをすることができます。
CRTの後ろのnの値によって、次のような設定が可能です。

nの値	モニターに表示する画面
0	テレビの映像を表示。（SHIFT ＋ テンキー部の =）
1	コンピュータの画面を表示。（SHIFT ＋ テンキー部の ・）
2	テレビ映像のコントラストを下げ、コンピュータの画面を重ねて表示。（SHIFT ＋ テンキー部の ＋）
3	テレビ映像とコンピュータの画面を同じコントラストで重ねて表示。

なお、高解像度ディスプレイモードでは、テレビ映像とコンピュータ画面の重ね表示（スーパーインポーズ）ができません。

参　照

『ユーザーズマニュアル』の「テレビコントロール」

2.14.3 CHANNEL

機　　能　テレビのチャンネルを変更します。

書　　式

```
CHANNEL n
```

n：チャンネル。１～１２の整数

省 略 形　CHAN.

文　　例　CHANNEL　１２

⇨１２チャンネルを設定します。

解　　説

テレビのチャンネルを変更します。

nは、放送局のチャンネルナンバーとは関係ありません。詳しくは、専用ディスプレイテレビの取扱説明書を参照してください。

チャンネルの設定は、SHIFT とテンキー部のキーを同時に押しても行なことができます。

CHANNEL 1	= SHIFT + 1	CHANNEL 7	= SHIFT + 7
CHANNEL 2	= SHIFT + 2	CHANNEL 8	= SHIFT + 8
CHANNEL 3	= SHIFT + 3	CHANNEL 9	= SHIFT + 9
CHANNEL 4	= SHIFT + 4	CHANNEL 1Ø	= SHIFT + /
CHANNEL 5	= SHIFT + 5	CHANNEL 11	= SHIFT + *
CHANNEL 6	= SHIFT + 6	CHANNEL 12	= SHIFT + −

〈注　　意〉　テレビ以外の映像または画面を表示しているときはチャンネルは変わりません。

参　　照　『ユーザーズマニュアル』の「テレビコントロール」

2.14

2.14.4　VOL

機　　能　テレビの音量を変更します。

書　　式

```
VOL〔n〕
```

n：変化の量。－62～64の整数。

文　　例　VOL－1∅
⇨テレビの音量が下がります。

解　　説　VOLは、専用ディスプレイテレビの音量を相対的に変化させるステートメントです。
nの値によって、音量が次のように変化します。

nの値	音　量　の　変　化
n＜∅	音量がダウンする。（SHIFT ＋ ⇩）
n＝∅	ミュートのon／offが反転する。（SHIFT ＋ O）※
n＞∅	音量がアップする。（SHIFT ＋ ⇧）
省　略	標準の音量。（SHIFT ＋ '）※

※）O、'はいずれもテンキー部のキーです。

参　　照　『ユーザーズマニュアル』の「テレビコントロール」

2.14

2.14.5 SCROLL

機　　能　　**コンピュータ画面のスクロールを行ないます。**

書　　式

```
SCROLL〔n〕
```

n：－3～3の整数。

省 略 形　　ＳＣＲＯ.

文　　例　　ＳＣＲＯＬＬ－１
⇨コンピュータ画面がゆっくり下へスクロールします。

解　　説　　テレビの映像画面とコンピュータ画面が重なった状態のとき（ＣＲＴ２かＣＲＴ３のとき）、コンピュータ画面の上下方向のスクロールを行ないます。
nの値によって、次のような設定ができます。

nの値	スクロールの速さと方向
3	上方向に速くスクロールする。
2	上方向にスクロールする。
1	上方向にゆっくりスクロールする。
0	スクロールの停止（＝ SHIFT ＋ BREAK）
－1	下方向にゆっくりスクロールする。
－2	下方向にスクロールする。
－3	下方向に速くスクロールする。
省　略	スクロールの停止。（＝ SHIFT ＋ BREAK）

画面のスクロール中にスクロールを停止するには、SHIFT ＋ BREAK キーを押してください。

参　　照　　『ユーザーズマニュアル』の「テレビコントロール」

2.14

2.15 カセット制御ステートメント

2.15.1 EJECT

機能　**カセットのふたを開けます。**

書式

```
EJECT
```

省略形　ＥＪ.

文例　ＥＪＥＣＴ
⇨専用データレコーダのカセットのふたを開けます。

解説　ＥＪＥＣＴを実行すると、専用データレコーダのカセットのふたを開けて、画面は入力待ちの状態となります。カセットがＦＡＳＴ、ＲＥＷの状態にあるときは、ＣＳＴＯＰが行なわれてから実行されます。

2.15.2 CSTOP

機能　**カセットテープの走行をストップします。**

書式

```
CSTOP
```

省略形　ＣＳＴ.

文例　ＣＳＴＯＰ
⇨カセットテープを停止状態にします。

解説　ＣＳＴＯＰは、専用データレコーダのカセットテープの走行をストップするためのステートメントです。
このステートメントは、ＦＡＳＴ、ＲＥＷにより、カセットが作動しているときに働き、ＬＯＡＤ、ＬＯＡＤＭ、ＳＡＶＥ、ＳＡＶＥＭ、ＶＥＲＩＦＹ、ＣＨＡＩＮ、ＭＥＲＧＥ、ＦＩＬＥＳ、ＬＦＩＬＥＳ、ＡＰＳＳの途中では実行しません。

2.15.3 FAST

機　能　カセットテープの早送りをします。

書　式

```
FAST
```

省略形　FA.

文　例　FAST
⇨カセットテープを早送りします。

解　説　FASTは、専用データレコーダのカセットテープの早送りをするステートメントです。カセットテープが入っていなければ何もしません。

2.15.4 REW

機　能　カセットテープの巻き戻しを行ないます。

書　式

```
REW
```

文　例　REW
⇨カセットテープの巻き戻しをします。

解　説　REWは、専用データレコーダのカセットテープの巻き戻しを行なうステートメントです。
カセットテープが入っていなければ何もしません。

2.15.5　ＡＰＳＳ

機　　能

カセットテープに記録されたプログラムの頭出しをします。

書　　式

```
APSS n
```

ｎ：－５０～５０の整数。

省略形

ＡＰ．

文　　例

ＡＰＳＳ　３

⇨カセットテープのプログラムで３ブロック目をＦＡＳＴ方向にて頭出しをします。

解　　説

ＡＰＳＳは、専用データレコーダのカセットテープに記録されているプログラムの頭出しをするステートメントです。

ｎ＝０のときは何もせず、正の数のときＦＡＳＴ方向の頭出しを、負の数のときＲＥＷ方向の頭出しをします。

途中で SHIFT ＋ BREAK キーを押すと、実行を止めることができます。

2.15

2.15.6 CMT

機　　能　**カセットの状態の設定を行ないます。**

書　　式

```
CMT=n
```

n=0～6、10。

省 略 形　CM.

文　　例　CMT=0

⇨カセットをEJECTの状態にします。(カセットのふたを開ける)

解　　説　CMTは、専用データレコーダのカセットの状態を設定するためのステートメントです。

nの値によって、次の設定をすることができます。

nの値	働　き	内　　　　容
0	EJECT	カセットのふたを開ける。
1	STOP	走行停止。
2	READ	再生状態にする。
3	FF	早送りする。
4	REW	巻戻す。
5	APSS+1	録音の切れ目まで早送りする。
6	APSS−1	録音の切れ目まで巻戻す。
10	WRITE	録音状態にする。

※2～10はカセットが入っていなければ何もしません。また、10はカセットテープのツメが折れていると何もしません。

カセットテープの消去は、

CMT=10

と設定して行ないます。

参　　照　3.8.6 CMT

2.15

2.16 サウンド制御ステートメント

2.16.1 BEEP

機　能

「ポッ」というビープ音を出します。

書　式

```
BEEP 〔n〕
```

n：0、1の整数。

省略形

BE.

文　例

BEEP
⇨「ポッ」という音を1回だけ出します。

解　説

BEEPは、「ポッ」というビープ音を出すためのステートメントです。
nの値によって、次のような設定ができます。

nの値	BEEPの動作
0	PSGのすべてのチャンネルの音を止めます。(BEEP1を解除します)
1	PSGのCチャンネルから「ピー」という連続音を出します。
省略	PSGのCチャンネルから「ポッ」という短音を出します。

PSG (Programmable Sound Generator) については、SOUNDステートメントを参照してください。

参　照

2.16.4 SOUND

サンプル プログラム

次の例は時刻を表示するプログラムで、秒が変わるたびにビープ音が鳴るようにしたものです。キーの入力があると停止します。

```
10 ' BEEP (例)
20 INIT:WIDTH40,25
30 T$=TIME$
40 LOCATE0,0
50 PRINT"時刻: ";TIME$
60 IF T$=TIME$ THEN 40
70 BEEP
80 K$=INKEY$(0)
90 IF K$="" THEN 30
100 END
```

30：TIME$の時刻をT$にコピーします。

40，50：TIME$の時刻を表示します。

60：TIME$が変化しない限り時刻表示を繰返しますが、TIME$が変化してT$と異なると70に実行が移ります。

70：ビープ音を鳴らします。

80：キーの入力を受付けます。

90：キーの入力がなければ30に戻り、あれば100に移ってプログラムの実行を終えます。

2.16.2 MUSIC／PLAY／MUSIC@／PLAY@

機　能　ミュージックを出します。

書　式

〔1〕MUSIC　x$
〔2〕PLAY　x$
〔3〕MUSIC@　x$
〔4〕PLAY@　x$

x$：楽譜を表わす文字式。単独の文字列、文字変数、配列、関数、およびそれらを連結したもの。

省略形

〔1〕MU.
〔2〕PL.
〔3〕MU.@
〔4〕PL.@

文　例

MUSIC "V9O4C1RCRDRERFRGRARBR+C3RGR+C"
⇨音階を出します。

解　説

MUSICおよびPLAYは、文字式x$で指定された楽譜に従ってミュージックを出すステートメントです。MUSIC@、PLAY@は、サウンド発生中でも次のステートメントを実行します。このステートメントを実行すると、CTCというLSIの管理下に入り、あるカウンタで、このステートメントの進行状況をみながら、次のステートメントを実行していきます。ただし、MUSIC、MUSIC@、PLAY、PLAY@が続く場合には、現在のPLAY@、MUSIC@が終了してから実行します。
MUSIC@、PLAY@はダイレクトモードでの実行はできません。

x$で表わされる文字列は、音名、オクターブ、音の長さ、音量、和音を指定し、以下に説明する文字から形成されます。

(1)　A～G、#………音名を表わします。指定文字と音名の対応は次に示す通りです。

指定文字	音　　名	
C	ド	ハ
＃C	ド＃	嬰ハ
D	レ	ニ
＃D	レ＃	嬰ニ（変ホ）
E	ミ	ホ
F	ファ	ヘ
＃F	ファ＃	嬰ヘ（変ト）
G	ソ	ト
＃G	ソ＃	嬰ト（変イ）
A	ラ	イ
＃A	ラ＃	嬰イ（変ロ）
B	シ	ロ

(2)　Ｏｎ、＋、－　………現在のオクターブを指定します。オクターブは８つあり、アルファベットのＯの後ろに１～８の番号をつけて表わします。初期状態はＯ４で、これは中央のＡ（４４０c／sec）を含むオクターブとなっています。

音名の前に＋をつけるとその音に限り１オクターブ上の音高を、－をつけると１オクターブ下の音高を表わします。

(3)　音名ｎ………音名の後ろに０～９の番号をつけて音の長さを指定します。番号と対応する音の長さは次の通りです。音名に番号がついていないときは、前の音と同じ長さが指定されます。

番　号	音の長さ	音符	相対値
0	３２分音符	𝅘𝅥𝅰	$\frac{1}{8}$
1	１６分音符	𝅘𝅥𝅯	$\frac{1}{4}$
2	付点１６分音符	𝅘𝅥𝅯.	$\frac{3}{8}$
3	８分音符	♪	$\frac{1}{2}$
4	付点８分音符	♪.	$\frac{3}{4}$
5	４分音符	♩	1
6	付点４分音符	♩.	$1\frac{1}{2}$
7	２分音符	𝅗𝅥	2
8	付点２分音符	𝅗𝅥.	3
9	全音符	𝅝	4

※相対値は、４分音符を１としたときの音符の長さです。

（４）Ｒｎ………Ｒは休符を表わす符合で、後ろに０～９の番号をつけて休みの長さを表わします。各番号に対応する休みの長さは、音の長さと同じです。

（５）Ｖｎ………Ｖは音量を表わす符合で、後ろに０～１５の番号をつけて大きさを指定します。この番号は、ＳＯＵＮＤステートメントの音量の指定データに一致しています。

（６）：（コロン）………コロンは、２重音や３重和音を出す場合の各声部を区切るための記号です。コロンを使うことにより、次のような使い方がで

きます。

（例）　"（チャンネルＡ）：（チャンネルＢ）"

"（チャンネルＡ）：（チャンネルＢ）：（チャンネルＣ）"

※ここで、（チャンネルＡ）は、チャンネルＡに対する文字列を意味しています。

テンポはＴＥＭＰＯまたはＰＬＡＹステートメントで指定します。

参照　2.16.3 ＴＥＭＰＯ／ＰＬＡＹ

サンプルプログラム

```
10 ' MUSIC/PLAY (例)
20 MUSIC"O5C7:O4E7:O3C7"
30 MUSIC"O5D7:O4F7:O3F7"
40 MUSIC"O5C7:O4E7:O3G7"
50 MUSIC"O4B7:O4D7:O2G7"
60 MUSIC"O5C7:O4E7:O3C7"
```

2.16.3 TEMPO／PLAY

機能

ミュージックのテンポを指定します。

書式

〔1〕TEMPO n
〔2〕PLAY n

n：30～7500。

省略形

〔1〕TE.
〔2〕PL.

文例

TEMPO 120
⇨MUSICのテンポを、1分間で120拍にします。

解説

MUSICやPLAYで書かれた曲のテンポを指定します。
nの値は、1分あたりの4分音符の拍数を表わします。
BASICの起動時、TEMPOは120に設定されています。

参照

2.16.2 MUSIC／PLAY／MUSIC@／PLAY@

サンプルプログラム

```
10 ' TEMPO <例>
20 INIT:WIDTH40,25
30 FOR I=0 TO 4
40 T=120*2^I
50 TEMPO T
60 LOCATE0,0:PRINT"TEMPO";T
70 MUSIC"O4G5"
80 MUSIC"O5C7D3FED"
90 MUSIC"O5G3RGRGAEF"
100 MUSIC"O5D3RDRDFED"
110 MUSIC"O5C3+CBAGFED"
120 MUSIC"O4R6  "
130 NEXT
```

70～120のMUSIC文をTEMPO120，TEMPO240，TEMPO480，TEMPO960，TEMPO1920の5つのテンポで順次実行します。
30～130：FOR－NEXTループ。

2.16.4 SOUND

機　能　音を出します。

書　式

```
〔1〕SOUND r, x
〔2〕SOUND r, x1, x2, ……
```

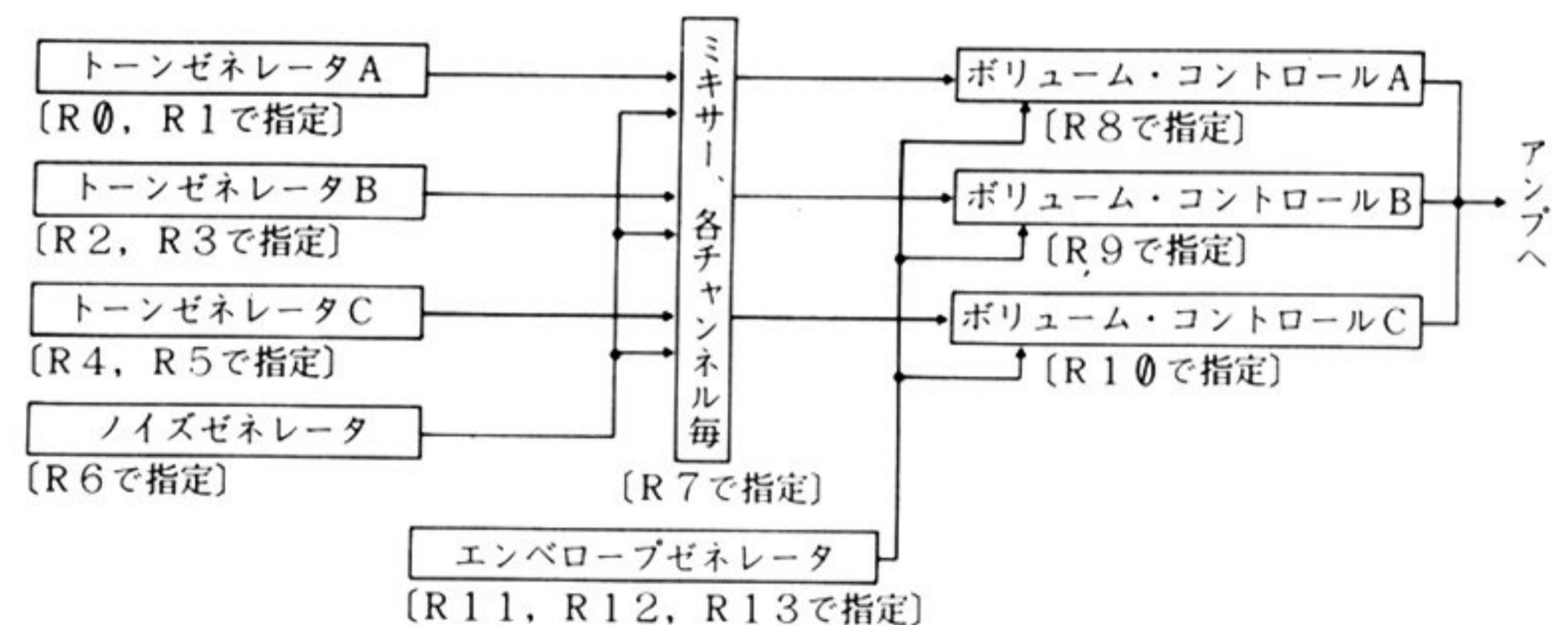
r：PSGのレジスタ番号。0～13整数。
x：0～255の整数を表わす数式。
x_1, x_2, ……：0～255の整数を表わす数式。

省略形　SO.

文　例　SOUND　7, &H38
⇨7番レジスタに&H38のデータを送ります。
SOUND　7, &H38, 13, 11, 12
⇨7番レジスタに&H38、8番に13、9番に11、10番に12のデータを送ります。

解　説　SOUNDは、音を出すためのステートメントで、PSG（Programmable Sound Generator）の14個のレジスタに直接データを書き込んで、さまざまな効果音を出すことができます。

書式〔2〕のようにデータをカンマ（,）で区切って並べて書くと、連続した複数のレジスタにそのデータを書き込むことができます。

本機に内蔵しているPSGは、3個のトーンゼネレータ、ノイズゼネレータ、ミキサー、3個のボリュームコントローラ、エンベロープゼネレータから構成されています。そのブロック図を下に示します。

（注）図中のR0～R13はレジスタ番号です。

次に１４個のレジスタの機能の一覧表を示します。

レジスタ番号	レジスタ機能	ビット構成 7 6 5 4 3 2 1 0	設定データ値（１５進）
0	チャンネルA周波数	下位８ビットＴ$_L$（微調）	0（高）～２５５（低）
1		上位４ビット Ｔ$_H$（粗調）	0（高）～　１５（低）
2	チャンネルB周波数	下位８ビットＴ$_L$（微調）	0（高）～２５５（低）
3		上位４ビット Ｔ$_H$（粗調）	0（高）～　１５（低）
4	チャンネルC周波数	下位８ビットＴ$_L$（微調）	0（高）～２５５（低）
5		上位４ビット Ｔ$_H$（粗調）	0（高）～　１５（低）
6	ノイズ周波数	５ビットＮ	0（高）～　３１（低）
7	チャンネル選択	IN/OUT: IOB IOA ／ ノイズ: C B A ／ トーン: C B A	0～６３ （各ビット共、0を設定すると選択される）
8	チャンネルA音量	M　４ビット	0（小）～１６（大）
9	チャンネルB音量	M　４ビット	0（小）～１６（大）
10	チャンネルC音量	M　４ビット	0（小）～１６（大）
11	エンベロープ周期	下位８ビットE$_L$（微調）	0（小）～２５５（大）
12		上位８ビットE$_H$（粗調）	0（小）～２５５（大）
13	エンベロープ形状	４ビット	0～１５

（レジスタ８～１０について）各チャンネルとも
M＝0のとき（データ値0～15）
4ビットの値で0から15段階に音量設定可能
M＝1のとき（データ値16～31）
音量はエンベロープに依存し4ビットデータは無効になる。

注）レジスタ番号７のビット７、６はＳＯＵＮＤ命令では使用しません。

サンプルプログラム

海岸の波の音、懐かしい蒸気機関車の走る音を出すプログラムです。
いずれも、実行中、キー入力があると停止するように作られています。

```
10 ' SOUND (例)
20 ' 波
30 SOUND6,20
40 SOUND7,&H37
50 SOUND8,16
60 SOUND11,0,90
70 SOUND13,14
80 A$=INKEY$(1)
90 SOUND7,&H3F
100 SOUND8,0
```

３0：ノイズの周波数をセットします。
４0：Aチャンネルのノイズだけ聴こえるようにセットします。
５0：Aチャンネルの音量がエンベロープに依存するようセットします。
６0：エンベロープ周期をセットします。
７0：エンベロープ波形をセットします。
８0：キーの入力を待ちます。
９0：すべてのチャンネルのゲートを閉じます。
１00：Aチャンネルの音量を0にします。

```
10 ' SOUND (例)
20 ' 蒸気機関車
30 SOUND0,45,2
40 SOUND6,30
50 SOUND7,&H2E
60 SOUND8,16,16
70 SOUND11,150,2
80 SOUND13,14
90 A$=INKEY$(1)
100 SOUND7,&H3F
110 SOUND8,0,0
```

３０：Ａチャンネルの周波数をセットします。

４０：ノイズの周波数をセットします。

５０：ＡチャンネルのトーンとＢチャンネルのノイズが聴こえるようにセットします。

６０：ＡチャンネルとＢチャンネルの音量がエンベロープに依存するようにセットします。

７０：エンベロープ周期をセットします。

８０：エンベロープ波形をセットします。

参　照　『ユーザーズマニュアル』の「ミュージック」

2.16.5 SOUND@

機　能　音を出します。

書　式

〔1〕**SOUND@ r, x**
〔2〕**SOUND@ r, x1, x2, ……**

r：PSGのレジスタ番号。0～13の整数。
x：0～65535の整数を表わす数式。
x1, x2, ……：0～65535の整数を示す数式。

省略形　SO.@

文　例

SOUND@0, &H4E2
⇨0番レジスタに&HE2、1番レジスタに&H4のデータを送ります。
SOUND@0, &H4E2, &H271
⇨0番レジスタに&HE2、1番に&H4、2番に&H71、3番に&H2のデータを送ります。

解　説

SOUND@は、SOUNDと同様、音をだすためのステートメントですが、連続した2つのレジスタにデータを書き込むときに用います。

これを実行すると、データの下位8ビットが最初のレジスタに、データの上位8ビットが次のレジスタに書き込まれます。

書式〔2〕のようにデータをカンマ（,）で区切って並べて書くと、複数個の連続したレジスタにデータを書き込むことができます。

参　照　2.16.4 SOUND

2.16.4 ＳＯＵＮＤのサンプルプログラム中の一部をＳＯＵＮＤ＠で書き換えた例を次に示します。

(例１)
```
10 ' SOUND@ (例1)
20 ' 波
30 SOUND6,20
40 SOUND7,&H37
50 SOUND8,16
60 SOUND@11,90*256
70 SOUND13,14
80 A$=INKEY$(1)
90 SOUND7,&H3F
100 SOUND8,0
```

(例２)
```
10 ' SOUND@ (例2)
20 ' 蒸気機関車
30 SOUND@0,2*256+45
40 SOUND6,30
50 SOUND7,&H2E
60 SOUND@8,16*256+16
70 SOUND@11,2*256+150
80 SOUND13,14
90 A$=INKEY$(1)
100 SOUND7,&H3F
110 SOUND@8,0
```

(例３)
```
10 ' SOUND@ (例3)
20 ' ﾍﾘｺﾌﾟﾀｰ
30 SOUND@0,20,30,9*256,&H3001,8*256+16,1
20*256+16,12*256+2
40 A$=INKEY$(1)
50 SOUND@7,&H3F,0
```

2.17 特殊ステートメント

2.17

2.17.1 MEM$

機能　**指定したメモリから任意の数だけ文字を取り出します。**

書式

〔1〕**MEM$ (a, n)**
〔2〕**MEM$ (a, n) =x $**

a：アドレス。0～&HFFFF。
n：取り出す文字の数。0～255の整数。
x $：文字式。"文字列"、文字変数、文字関数、およびそれらを連結したもの。

文例　D$=MEM$（&HC000, 10）
⇨メモリのアドレス&HC000から始まるエリアから10個の文字を取り出してD$に代入します。

解説　メモリ内の指定したアドレスaから始まる、n字を取り出します。
〔2〕の書式で使うと、逆に、メモリの指定されたアドレスにx $の表わす文字列を書き込むことができます。

参照　3.6.2 PEEK、2.3.12 POKE

サンプルプログラム　次の例は、MEM$ステートメントでE000番地以降に文字列を書き込んだ後、MEM$関数で1文字ずつ取り出すプログラムです。

```
10 ' MEM$ <例>
20 CLEAR &HE000
30 INPUT"文字列=",X$
40 L=LEN(X$)
50 MEM$(&HE000,L)=X$
60 FOR I=0 TO L-1
70 ADR=&HE000+I
80 A$=MEM$(ADR,1)
90 PRINT HEX$(ADR);": ";A$
100 NEXT
110 END
```

30：入力した文字列をX$に入れます。
50：X$の文字列をE000番地以降に書き込みます。
60～100：書き込んだ文字列を1文字ずつ読み込んで表示します。

2.17

```
RUN
文字列=Personal Computer
E000: P
E001: e
E002: r
E003: s
E004: o
E005: n
E006: a
E007: l
E008:
E009: C
E00A: o
E00B: m
E00C: p
E00D: u
E00E: t
E00F: e
E010: r
Ok
```

2.17.2 PAUSE

機　能　**プログラムの実行の一時停止をします。**

書　式

```
PAUSE n
```

n：停止時間の長さ。0～255の整数。(0．1秒単位)

省略形　PA.

文　例　PAUSE　100

⇨プログラムの実行を約10秒間休止します。

解　説　プログラムがPAUSEステートメントのところに来ると、一時停止し、指定時間が経過すると、次に進みます。

停止時間は、1単位につき約0．1秒で、最大約25秒間の停止が可能です。

参　照　2.17.3 WAIT

サンプルプログラム

```
10 ' PAUSE <例>
20 T=10
30 TIME=0
40 PAUSE T
50 PRINT"PAUSE";T;"= 約";TIME;"秒
60 T=T+10
70 IF T<100 THEN 30
80 END
```

上の例は、PAUSE10からPAUSE90までの停止時間を計るプログラムです。

20：T＝10の初期値設定。

30：TIMEを0にセットします。

40：T単位の一時停止をします。

50：経過時間TIMEを表示します。

70：T<100ならば30番へ戻ります。

```
RUN
PAUSE 10 : 時間  約 1 秒
PAUSE 20 : 時間  約 2 秒
PAUSE 30 : 時間  約 3 秒
PAUSE 40 : 時間  約 4 秒
PAUSE 50 : 時間  約 5 秒
PAUSE 60 : 時間  約 6 秒
PAUSE 70 : 時間  約 7 秒
PAUSE 80 : 時間  約 8 秒
PAUSE 90 : 時間  約 9 秒
Ok
```

2.17.3 WAIT

2.17

機能　入力ポートが指定された状態になるまで待ち続けます。

書式

```
WAIT n, x1 〔, x2〕
```

n：入力ポートの番号。0～&HFFFF。

x_1，x_2：数式。単独の定数、数値変数でもかまわない。0～255の整数を値にもつこと。

省略形　WA.

文例　WAIT&H3000+40*24，&HFF，&H46

⇨WIDTH40，25の画面において SHIFT キーが押されるまで（画面左下隅の文字がFからLに変わるまで）待ち続けます。

解説　WAITステートメントにより、指定された入力ポートが特定のビット・パターンをもつデータになるまで、次の実行に移らないようにできます。

入力ポートで読み込んだデータは、x_2 とXOR[1] され、次に x_1 とAND[2] されます。その結果が0の場合は、次のデータを入力ポートから読み込んで再び同じ操作を行ないます。こうして、読み込んだデータに対してXORとANDをとる操作を繰り返し、結果が0でなくなったときにはじめてWAITの次のステートメントに進みます。

1）XOR　EXCLUSIVE　OR（1.9「演算子」参照）

2）AND（1.9「演算子」参照）

（例）WAIT　&H1A01，8，&HFF

いま読み込んだデータの値が&H39のとき、まず&HFFとXORをとって、

&H39　XOR　&HFF　=　&HC6

次にこの結果と8のANDをとって、

&HC6　AND　8　=　0

したがって、WAITはポートからの次のデータを読み込みにかかります。

このようにして、読み込んだデータの値が&HB1のとき、

&HB1　XOR　&HFF　=　&H4E

この結果と8のANDをとって、

&H4E　AND　8　=　8

となり、WAITの次のステートメントに実行が移ります。　WAITステートメントにより、入力されたデータのビットパターンをテストすることができます。

テストしたいビットの位置は、x_1 のビットパターンの対応位置に1を設定することで指定できます。

x_2 を省略すると、入力ポートのデータの各ビットが1かどうかテストすることができます。

x_2 を指定すると、入力ポートのデータのビットが0かどうかテストすることができます。このとき、x_1, x_2 のビットパターンのうち、テストしたビット位置に1をセットしておきます。

参照　2.17.2 PAUSE

3章

関数・システム変数

3.1 数値関数

3.1.1 SIN

3.1

機能　三角関数のサインを返します。

書式

```
SIN (x)
```

x：サインを求めたい数式。単独の定数、数値変数を含む。ラジアンで表わした角度。

文例　PRINT SIN (RAD(45))
⇨sin45°の値 .70710678を表示します。

解説　数式xのサイン、「sin x」が、この関数の値になります。
数式xは、ラジアン単位の角度を表わします。単独の数値定数、数値変数でもかまいません。
xが度単位の角度の場合、文例のようにラジアン単位に変換しなければなりません。度をラジアンに変換する方法には、RAD関数を使う方法と、PAI関数を使ってPAI（1）／180またはπ／180をかける方法の2通りあります。
関数の値は、単精度で有効数字8けたが保証されています。ただし、xが倍精度型実数か倍精度型実数を含むときは、倍精度、有効数字16けたとなります。

参照　3.1.2 COS、3.1.3 TAN、3.1.5 RAD、3.1.6 PAI

サンプルプログラム

```
10 'SIN  COS  TAN  (例)
20 PRINT"(度) SIN            COS             TAN"
30 FOR X=0 TO 90 STEP 15
40 PRINT X;
50 PRINT TAB(5) SIN(RAD(X));TAB(20) COS(RAD(X));
60 IF X=90 THEN 80
70 PRINT TAB(35) TAN(RAD(X));
80 PRINT
90 NEXT X
```

30～90：SINX、COSX、TANXの値をX＝0°からX＝90°まで15°おきに求めます。

```
RUN
(度) SIN            COS             TAN
 0   0              1               0
 15  .25881904      .96592583       .26794919
 30  .5             .8660254        .57735027
 45  .70710678      .70710678       1
 60  .8660254       .5              1.7320508
 75  .96592583      .25881905       3.7320508
 90  1              4.6566129E-10
```

3.1.2 COS

機　能

三角関数のコサインを返します。

書　式

```
COS (x)
```

x：コサインを求めたい数式。単独の定数、数値変数を含む。ラジアンで表わした角度。

文　例

PRINT　COS (RAD (3 0))
⇨COS 3 0°の値 .8 6 6 0 2 5 4を表示します。

解　説

数式 x のコサイン、「cos　x」がこの関数の値になります。
数式 x は、ラジアン単位の角度を表わします。単独の数値定数、数値変数でもかまいません。
x が度単位の角度の場合、文例のようにラジアン単位に変換しなければなりません。度をラジアンに変換する方法には、RAD関数を使う方法と、PAI関数を使ってPAI (1) ／1 8 0または π／1 8 0をかける方法の2通りあります。
関数の値は、単精度で有効数字8けたが保証されています。ただし、x が倍精度型実数か倍精度型実数を含むときは、倍精度、有効数字1 6けたとなります。

参　照

3.1.1 SIN、3.1.3 TAN、3.1.5 RAD、3.1.6 PAI

サンプルプログラム

3.1.1 SINのサンプルプログラム参照。

3.1.3　TAN

機　能

三角関数のタンジェントを返します。

書　式

```
TAN (x)
```

x：タンジェントを求めたい数式。単独の定数、数値変数を含む。ラジアンで表わした角度。

文　例

PRINT　TAN（RAD（45））
⇨tan45°の値　1を表示します。

解　説

数式xのタンジェント、「tan　x」が、この関数の値になります。
数式xは、ラジアン単位の角度を表わします。単独の数値定数、数値変数でもかまいません。

xが度単位の角度の場合、文例のようにラジアン単位に変換しなければなりません。度をラジアンに変換する方法には、RAD関数を使う方法と、PAI関数を使ってPAI（1）／180またはπ／180をかける方法の2通りあります。

関数の値は、単精度で有効数字8けたが保証されています。ただし、xが倍精度型実数か倍精度型実数を含むときは、倍精度、有効数字16けたとなります。

参　照

3.1.1 SIN、3.1.2 COS、3.1.5 RAD、3.1.6 PAI

サンプルプログラム

3.1.1 SINのサンプルプログラム参照。

3.1.4　ＡＴＮ

機　能

アークタンジェントを返します。

書　式

```
ATN (x)
```

x：アークタンジェントを求めたい数式。単独の定数、数値変数を含む。

文　例

A＝ATN（X）

⇨Xの値のアークタンジェントがAに代入されます。

解　説

数式xのアークタンジェント（逆正接）、「arctan　x」が、この関数の値になります。

数式xは、単独の数値定数、数値変数でもかまいません。

関数の値はラジアン単位で、－π／２～π／２（－１．５７０７９６３～１．５７０７９６３）の範囲にあり、単精度、有効数字８けたが保証されています。ただし、xが倍精度型実数か、倍精度型実数を含むときは、倍精度、有効数字１６けたとなります。

参　照

3.1.1 ＳＩＮ、3.1.2 ＣＯＳ、3.1.5 ＲＡＤ、3.1.6 ＰＡＩ

サンプルプログラム

```
10 ' ATN (例)
20 PRINT " x          ATN(x)      (度)
30 FOR X=-10 TO 10 STEP 2
40 Y=X/10
50 PRINT Y;TAB(5);
60 PRINTUSING"#####.####";ATN(Y);ATN(Y)*180/PAI(1)
70 NEXT
```

３０～７０：ＡＴＮ(ｘ)の値をｘ＝－１からｘ＝１まで０．２おきに求めています。

６０：ＡＴＮ(ｘ) の値とそれに$\frac{180}{\pi}$をかけて度単位に変換した値を求めて表示します。

```
RUN
 x          ATN(x)     (度)
-1          -.7854   -45.0000
-.8         -.6747   -38.6598
-.6         -.5404   -30.9638
-.4         -.3805   -21.8014
-.2         -.1974   -11.3099
 0          0.0000     0.0000
 .2         0.1974    11.3099
 .4         0.3805    21.8014
 .6         0.5404    30.9638
 .8         0.6747    38.6598
 1          0.7854    45.0000
Ok
```

3.1.5 RAD

3.1

機能　数値を度単位からラジアン単位に変換します。

書式

```
RAD (x)
```

x：変換したい数式。単独の定数、数値変数でもかまいません。度単位。

文例　A＝RAD（3０）

⇨3０°をπ／18０倍して、ラジアン単位に変換した値をAに代入します。

解説　度単位の数式xをπ／18０倍して、ラジアン単位に変換します。

この関数は、三角関数SIN、COS、TANの値を求めるときに使います。

関数の値は、単精度で有効数字8けたが保証されています。ただし、xが倍精度型実数か倍精度型実数を含むときは、倍精度、有効数字16けたとなります。

参照　3.1.1 SIN、3.1.2 COS、3.1.3 TAN

サンプルプログラム

```
10 ' RAD (例)
20 PRINT"(度)     SIN       COS"
30 FOR X=0 TO 90 STEP 15
40 PRINTUSING"###";X;
50 PRINTUSING"####.####";SIN(RAD(X));COS(RAD(X))
60 NEXT
```

3０～6０：SINXとCOSXの値をX＝０°からX＝9０°まで15°おきに求めます。

5０：SINとCOSを求めるときはRADを使ってXをラジアン単位にします。

```
RUN
(度)     SIN       COS
  0    0.0000    1.0000
 15    0.2588    0.9659
 30    0.5000    0.8660
 45    0.7071    0.7071
 60    0.8660    0.5000
 75    0.9659    0.2588
 90    1.0000    0.0000
Ok
```

3.1.6 PAI

機　能　円周率πのx倍の値を返します。

書　式

```
PAI (x)
```

x：数式。単独の数値定数、数値変数でもかまいません。

文　例　A＝PAI（1）

⇨円周率πの値3.1415927がAに代入されます。

解　説　円周率π≒3.1415927のx倍が、この関数の値となります。

関数の値は、単精度で有効数字8けたが保証されています。ただし、xが倍精度型実数か倍精度型実数を含むときは、倍精度、有効数字16けたとなります。

〈注　意〉　PAI（1）の代わりに、特殊文字のπ（パイ）を使用することができます。

πの値は3.1415927の単精度ですが、π#の倍精度型の変数とすると、3.141592653589793の値をもちます。

πは GRAPH ＋ A キーを押して表示することができます。

サンプルプログラム　球の体積と表面積を求めるプログラムを作ってみましょう。

球の体積は、

$$V=\frac{4}{3}\pi r^3$$

球の表面積は、$S=4\pi r^2$

で求められます。

```
10 ' PAI (例)
20 INPUT "半径(cm)=";R
30 V=PAI(4)*R^3/3
40 S=PAI(4)*R^2
50 PRINT " 体積   =";V;"(cm3)"
60 PRINT " 表面積 =";S;"(cm2)"

RUN
半径(cm)=? 10
 体積   = 4188.7902 (cm3)
 表面積 = 1256.6371 (cm2)
Ok
```

20：球の半径Rの値を入力します。

30：球の体積を計算してVに代入します。

40：球の表面積を計算してSに代入します。

50，60：体積Vと表面積Sを表示します。

3.1.7 ABS

3.1

機能

数式の絶対値を返します。

書式

```
ABS (x)
```

x：絶対値を求める数式。単独の数値定数、数値変数でもかまいません。

文例

PRINT ABS (−23)

⇨−23の絶対値23が表示されます。

解説

数式 x の絶対値｜x｜、この関数の値になります。
すなわち、ABSは、x の値から符号を取り去った値を返します。

関数の値は、単精度で有効数字8けたが保証されていますが、x が倍精度型実数か、倍精度型実数を含むときは、倍精度、有効数字16けたとなります。

サンプルプログラム

```
10 ' ABS (例)
20 PRINT"数値            絶対値"
30 X=-2.5
40 FOR I=1 TO 10
50 PRINT X,ABS(X)
60 X=X+.5
70 NEXT


RUN
数値          絶対値
-2.5           2.5
-2             2
-1.5           1.5
-1             1
-.5            .5
 0             0
 .5            .5
 1             1
 1.5           1.5
 2             2
Ok
```

3.1.8 SGN

機能　数式の符号を返します。

書式

```
SGN (x)
```

x：符号を調べたい数式。単独の数値定数、数値変数でもかまいません。

文例　ＰＲＩＮＴ　ＳＧＮ（－２３）
⇨－２３は負の数なので、－１が表示されます。

解説　数式ｘの値によって、次の値がこの関数の値になります。

・x＞0　のとき　1、
・x＝0　のとき　0、
・x＜0　のとき－1、

ｘは、数値型であれば、どんな値でもかまいません。

ＳＧＮ関数は、ＳＩＮ関数と同じ「サイン」と発音しますが、全く別の働きをもつものなので注意してください。

サンプルプログラム

```
10 ' SGN (例)
20 PRINT"数値         符号"
30 X=-2.5
40 FOR I=1 TO 10
50 PRINT X,SGN(X)
60 X=X+.5
70 NEXT

RUN
数値        符号
-2.5         -1
-2           -1
-1.5         -1
-1           -1
-.5          -1
 0            0
 .5           1
 1            1
 1.5          1
 2            1
Ok
```

3.1.9 INT

機　能

数式の値を越えない最大の整数を返します。

書　式

```
INT (x)
```

x：数式。単独の数値定数、数値変数でもかまいません。

書　式

PRINT　INT (－3.65)

⇨－3.65を越えない最大の整数－4が表示されます。

解　説

数式xの値を越えない最大の整数〔x〕が、この関数の値になります。

(例) INT (3.65) ……この値は、3となります。

INT (－3.65) ……この値は、－4となります。

参考のためにCINT、FIXの返す値を示しておきます。

CINT (3.65) ……値は4　　FIX (3.65) ……値は3

CINT (－3.65) …値は－4　　FIX (－3.65) …値は－3

xは、数値型の数式で、単独の数値定数、数値変数でもかまいません。

参　照

3.1.12 CINT、3.1.10 FIX

サンプルプログラム

```
10 ' INT (例)
20 PRINT"  x       INT(x)"
30 X=-3.14
40 FOR I=1 TO 10
50 PRINT X,INT(X)
60 X=X+.75
70 NEXT

RUN
  x        INT(x)
-3.14       -4
-2.39       -3
-1.64       -2
-.89        -1
-.14        -1
 .61         0
 1.36        1
 2.11        2
 2.86        2
 3.61        3
Ok
```

3.1.10 ＦＩＸ

機　　能

数式の値の整数部を返します。

書　　式

```
FIX (x)
```

x：数式。単独の数値定数、数値変数でもかまいません。

文　　例

ＰＲＩＮＴ　ＦＩＸ（３.６５）
⇨３.６５の整数部３が表示されます。

解　　説

数式ｘの値の整数部のみ取り出して、小数点以下を切り捨てた値が、この関数の値になります。

（例）ＦＩＸ（３.６５）……この値は、３となります。
　　　ＦＩＸ（－３.６５）……この値は、－３となります。

参考のためにＩＮＴ、ＣＩＮＴが返す値を示しておきます。

ＩＮＴ（３.６５）……値は３　　ＣＩＮＴ（３.６５）……値は４
ＩＮＴ（－３.６５）…値は－４　　ＣＩＮＴ（－３.６５）…値は－４

ｘは、数値型の数式で、単独の数値定数、数値変数でもかまいません。

参　　照

3.1.11 ＦＲＡＣ、3.1.9 ＩＮＴ、3.1.12 ＣＩＮＴ

サンプル プログラム

```
10 ' FIX (例)
20 PRINT"  x       FIX(x)"
30 X=-3.14
40 FOR I=1 TO 10
50 PRINT X,FIX(X)
60 X=X+.75
70 NEXT

RUN
  x       FIX(x)
-3.14      -3
-2.39      -2
-1.64      -1
-.89        0
-.14        0
 .61        0
 1.36       1
 2.11       2
 2.86       2
 3.61       3
Ok
```

3.1.11 FRAC

機　能　数式の値の小数部を返します。

書　式

```
FRAC (x)
```

x：数式。単独の数値定数、数値変数でもかまいません。

解　説　数式xの値の小数部のみ取り出して、整数部を取り去った値が、この関数の値になります。

（例）FRAC（3.65）……この値は、.65となります。
　　　FRAC（－3.65）……この値は、－.65となります。
　　　　※.65＝0.65、　－.65＝－0.65です。

xは、数値型の数式で、単独の数値定数、数値変数でもかまいません。

参　照　3.1.10 FIX

サンプルプログラム

```
10 ' FRAC (例)
20 INPUT"x=";X
30 PRINT"FRAC(x)=";FRAC(X)
40 PRINT"FIX(x) =";FIX(X)
50 PRINT"FRAC(x)+FIX(x)=";FRAC(X)+FIX(X)
```

20：変数Xの値を入力します。
30：Xの小数部を求めます。
40：Xの整数部を求めます。
50：小数部と整数部をたしてもとのXの値に戻します。

```
RUN
x=? 3.141592
FRAC(x)= .141592
FIX(x) = 3
FRAC(x)+FIX(x)= 3.141592
Ok
```

3.1.12 CINT

機　　能

数式の値を整数型に変換します。

書　　式

```
CINT (x)
```

x：型変換を行ないたい数式。単独の数値定数、数値変数でもかまいません。

文　　例

I＝CINT（23.54）
⇨23.54の値を四捨五入した24をIに代入します。

解　　説

数式xの値の小数第1位を四捨五入して、整数型に変換します。

xは、単独の数値定数、数値変数でもかまいませんが、どのような形にせよ、値が−32768〜32767の範囲を越えてはなりません。

整数を返す関数として、ほかにFIXとINTがありますが、いずれも働きが違いますので、使い方に注意してください。

（例）CINT（3.65）……四捨五入して、値は4
　　　CINT（−3.65）…四捨五入して、値は−4
　　　FIX（3.65）………整数部3が値
　　　FIX（−3.65）……整数部−3が値
　　　INT（3.65）………越えない整数3が値
　　　INT（−3.65）……越えない整数−4が値

参　　照

3.1.13 CSNG、3.1.14 CDBL、3.1.10 FIX、3.1.9 INT

サンプルプログラム

```
10 ' CINT (例)
20 PRINT"  x              CINT(x)"
30 A%=3
40 B!=3.14
50 C#=3.141592653#
60 D!=4.898
70 E#=-4.898979485#
80 PRINT A%;TAB(15);CINT(A%)
90 PRINT B!;TAB(15);CINT(B!)
100 PRINT C#;TAB(15);CINT(C#)
110 PRINT D!;TAB(15);CINT(D!)
120 PRINT E#;TAB(15);CINT(E#)
```

30：整数型変数A％に3を代入します。
40：単精度型変数B！に3.14を代入します。
50：倍精度型変数C＃に3.141592653を代入します。
60：単精度型変数D！に4.898を代入します。
70：倍精度型変数E＃に−4.898979485を代入します。
80：A％の値とCINT（A％）の値を表示します。
90：B！の値とCINT（B！）の値を表示します。
100：C＃の値とCINT（C＃）の値を表示します。
110：D！の値とCINT（D！）の値を表示します。
120：E＃の値とCINT（E＃）の値を表示します。

```
RUN
  x             CINT(x)
 3                3
 3.14             3
 3.141592653      3
 4.898            5
-4.898979485     -5
Ok
```

3.1

3.1.13 CSNG

機能

数式の値を単精度型に変換します。

書式

```
CSNG (x)
```

x：型変換を行ないたい数式。単独の数値定数、数値変数でもかまいません。

文例

S＝CSNG（1.23456789Ø1#）

⇨1.23456789Ø1#を単精度に変換した1.23456789をSに代入します。

解説

数式xの値を単精度型に変換します。

xは、単独の数値定数、数値変数でもかまいません。

xの値が整数のときは、ほとんど意味がありませんが、倍精度型実数のときは、有効数字8けたの単精度型に変換されます。

参照

『ユーザーズマニュアル』の付録「数値精度の変換」、3.1.12 CINT、3.1.14 CDBL

サンプルプログラム

```
10 ' CSNG (例)
20 PRINT"    x              CSNG(x)"
30 A%=3
40 B!=3.14
50 C#=3.141592653#
60 D#=4.898979485566357#
70 PRINT A%;TAB(15);CSNG(A%)
80 PRINT B!;TAB(15);CSNG(B!)
90 PRINT C#;TAB(15);CSNG(C#)
100 PRINT D#;TAB(15);CSNG(D#)
```

3Ø：整数型変数A%に3を代入します。

4Ø：単精度型変数B！に3.14を代入します。

5Ø：倍精度型変数C#に3.141592653を代入します。

6Ø：倍精度型変数D#に4.898979485566357を代入します。

7Ø：A%の値とCSNG(A%)の値を表示します。

8Ø：B！の値とCSNG(B！)の値を表示します。

9Ø：C#の値とCSNG(C#)の値を表示します。

1ØØ：D#の値とCSNG(D#)の値を表示します。

```
RUN
   x              CSNG(x)
 3                3
 3.14             3.14
 3.141592653      3.1415927
 4.898979485566357   4.8989795
Ok
```

3.1.14 CDBL

機能

数式の値を倍精度型に変換します。

書式

```
CDBL (x)
```

x：型変換を行ないたい数式。単独の数値定数、数値変数でもかまいません。

文例

D＃＝CDBL（2.3）

⇨2.3を倍精度に変換した2.2999999998137 35をD＃に代入します。

解説

数式xの値を倍精度型に変換します。

xは、単独の数値定数、数値変換でもかまいません。

xの値が整数のときは、ほとんど意味がありませんが、単精度型実数のときは、有効数字16けたの倍精度型に変換されます。

ただし、変換されて有効数字のけた数が2倍になっても、数値の精度は上がりません。

参照

『ユーザーズマニュアル』の付録「数値精度の変換」、3.1.12 CINT、3.1.13 CSNG

サンプルプログラム

```
10 ' CDBL (例)
20 PRINT"  x";SPC(15);"CDBL(x)"
30 A%=3
40 B!=3.14
50 C#=3.141592653#
60 D!=4.8989
70 PRINT A%;TAB(15);CDBL(A%)
80 PRINT B!;TAB(15);CDBL(B!)
90 PRINT C#;TAB(15);CDBL(C#)
100 PRINT D!;TAB(15);CDBL(D!)
```

30：整数型変数A％に3を代入します。

40：単精度型変数B！に3.14を代入します。

50：変数C＃に3.141592653を代入します。

60：単精度型変数D！に4.8989を代入します。

70：A％の値とCDBL（A％）の値を表示します。

80：B！の値とCDBL（B！）の値を表示します。

90：C＃の値とCDBL（C＃）の値を表示します。

100：D！の値とCDBL（D！）の値を表示します。

```
RUN
  x             CDBL(x)
 3             3
 3.14          3.139999999664724
 3.141592653   3.141592653
 4.8989        4.898900000037849
Ok
```

3.1.15 ＥＸＰ

機　能　**指数関数の値を返します。**

書　式

```
EXP (x)
```

ｘ：指数関数を求める数式。単独の数値定数、数値変数でもかまいません。

文　例　ＰＲＩＮＴ　ＥＸＰ（２）

⇨自然対数の底ｅの２乗の値７．３８９０５６１を表示します。

解　説

自然対数の底ｅ≒２．７１８２８１８のｘ乗が、この関数の値となります。

関数の値は、単精度で有効数字８けたが保証されていますが、ｘが倍精度型実数か、倍精度型実数を含むときは、倍精度、有効数字１６けたの値となります。

ｘの値が８８．０２９６９２以上大きいと、関数の値が大きくなりすぎるので、「Ｏｖｅｒｆｌｏｗ」のエラーが出ます。

サンプルプログラム

（例１）

```
10 ' EXP (例1)
20 INPUT "x=";X
30 PRINT "EXP(x)=";EXP(X)
```

２０：Ｘの値を入力します。

３０：ＥＸＰ（ｘ）の値を表示します。

```
RUN
x=? 2
EXP(x)= 7.3890561
Ok
```

（例２）

指数関数ＥＸＰを使ってｓｉｎｈ ｘとｃｏｓｈ ｘ関数を計算してみましょう。

$$\sinh x = \frac{e^{x} - e^{-x}}{2},\quad \cosh x = \frac{e^{x} + e^{-x}}{2}$$

プログラムは次のようになります。

```
10 ' EXP (例2)
20 INPUT "x=";X
30 PRINT "EXP(x) =";EXP(X)
40 PRINT "sinh(x)=";(EXP(X)-EXP(-X))/2
50 PRINT "cosh(x)=";(EXP(X)+EXP(-X))/2
```

２０：Ｘの値を入力します。

３０：ＥＸＰ（ｘ）の値を表示します。

４０：ｓｉｎｈ ｘの値を求めて表示します。

５０：ｃｏｓｈ ｘの値を求めて表示します。

```
RUN
x=? 2
EXP(x) = 7.3890561
sinh(x)= 3.6268604
cosh(x)= 3.7621957
Ok
```

3.1.16 LOG

機能 自然対数を返します。

書式

```
LOG (x)
```

x：数式。単独の数値定数、数値変数でもかまいません。

文例 PRINT LOG (2)

⇨自然対数$\log_e 2$の値、.69314718を表示します。

解説 数式の自然対数ln x（$=\log_e x$）が、この関数の値となります。

$\log_B A$は、$\log_B A = \log_e A / \log_e B$の式より、

LOG (A) ／LOG (B)

で求めることができます。ただし、B＞0、B≠1。

常用対数を求めるときは、$\log_{10} X = \log_e X / \log_e 10$の式より、

LOG (X) ／LOG (10)

で求めることができます。

LOG関数の値は、単精度で有効数字8けたが保証されていますが、xが倍精度型実数か倍精度型実数を含むときは、倍精度、有効数字16けたとなります。

参照 3.1.15 EXP

サンプルプログラム

(例1) 自然対数と常用対数を求めるプログラムを示します。

```
10 ' LOG (例1)
20 INPUT" x=";X
30 PRINT"LOG(x)=";LOG(X)
40 PRINT"LOG 10(x)=";LOG(X)/LOG(10)
```

20：変数Xの値を入力します。

30：自然対数LOG(x)を求めて表示します。

40：常用対数LOG(x)/LOG(10)を求めて表示します。

```
RUN
 x=? 2
LOG(x)= .69314718
LOG 10(x)= .30103
Ok
```

(例2) 公式 ln A＋ln B＝ln AB

ln A－ln B＝ln (A/B)

を実際に確かめてみましょう。

```
10 ' LOG (例2)
20 INPUT" a=";A
30 INPUT" b=";B
40 PRINT"LOG(a)=";LOG(A)
50 PRINT"LOG(b)=";LOG(B)
60 PRINT"LOG(a)+LOG(b)=";LOG(A)+LOG(B)
70 PRINT"LOG(a*b)      =";LOG(A*B)
80 PRINT"LOG(a)-LOG(b)=";LOG(A)-LOG(B)
90 PRINT"LOG(a/b)      =";LOG(A/B)
```

２０，３０：ＡとＢの値を入力します。

４０，５０：ＬＯＧ(Ａ)とＬＯＧ(Ｂ)の値を求めて表示します。

６０：ＬＯＧ(Ａ)＋ＬＯＧ(Ｂ)の値を求めて表示します。

７０：ＬＯＧ(Ａ＊Ｂ)の値を求めて表示します。

８０：ＬＯＧ(Ａ)－ＬＯＧ(Ｂ)の値を求めて表示します。

９０：ＬＯＧ(Ａ/Ｂ)の値を求めて表示します。

```
RUN
 a=? 3
 b=? 2
LOG(a)= 1.0986123
LOG(b)= .69314718
LOG(a)+LOG(b)= 1.7917595
LOG(a*b)      = 1.7917595
LOG(a)-LOG(b)= .40546511
LOG(a/b)      = .40546511
Ok
```

3.1.17 SQR

3.1

機　能　平方根を返します。

書　式

```
SQR (x)
```

x：数式。単独の数値定数、数値変数でもかまいません。

文　例　ＰＲＩＮＴ　ＳＱＲ（3）

⇨3の平方根1.7320508を表示します。

解　説　数式xの平方根が、この関数の値になります。

xは単独の数値定数、数値変数でもかまいませんが、0以上の値でなければなりません。

関数の値は、単精度で有効数字8けたが保証されていますが、xが倍精度型実数か、倍精度型実数を含むときは、倍精度、有効数字16けたとなります。

サンプルプログラム

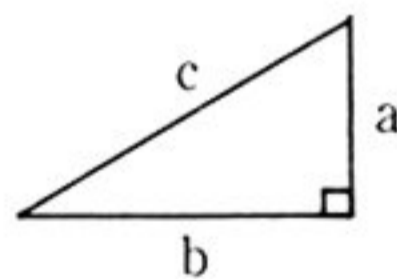
（例1）

```
10 ' SQR (例1)
20 INPUT"x=";X
30 PRINT"SQR(x) =";SQR(X)
```

20：変数Xの値を入力します。

30：SQR(x)の値を表示します。

```
RUN
x=? 5
SQR(x) = 2.236068
Ok
```

（例2）　ピタゴラスの定理を利用して、直角三角形の2辺a，bから残りの長さcを求めるプログラムです。

```
10 ' SQR (例2)
20 INPUT "a=";A
30 INPUT "b=";B
40 PRINT "c=";SQR(A*A+B*B);SQR(A^2+B^2)
```

20：変数Aの値を入力します。

30：変数Bの値を入力します。

40：$\sqrt{A*A+B*B}$ と $\sqrt{A^2+B^2}$ の2通りの計算式の値を表示します。

```
RUN
a=? 2
b=? 3
c= 3.6055513  3.6055513
Ok
```

3.1.18 ＦＡＣ

機　能　**数式の値の階乗を返します。**

書　式

```
FAC (n)
```

ｎ：数式。単独の数値定数、数値変数でもかまいません。

文　例　ＰＲＩＮＴ　ＦＡＣ（５）

⇨５の階乗（５！）の値１２０が表示されます。

解　説

数式ｎの階乗ｎ！が、この関数の値となります。

ｎは、単独の数値定数、数値変数でもかまいませんが、値が０以上３３以下でなければなりません。

ｎが、単精度型、倍精度型の実数のときは、自動的に小数点以下が切り捨てられて、関数の値が求められます。

サンプルプログラム

```
10 ' FAC (例)
20 INPUT "n=";N
30 PRINT "n=";N,"FAC(n)=";FAC(N)
```

２０：変数Ｎの値を入力します。

３０：Ｎのの値とＦＡＣ(Ｎ)の値を表示します。

```
RUN
n=? 10
n= 10         FAC(n)= 3628800
Ok
```

3.1.19　SUM

機　　能

数式nの $\sum_{i=1}^{n} i$ （＝1＋2＋……＋n）を返します。

書　　式

```
SUM (n)
```

n：数式。単独の数値定数、数値変数でもかまいません。

文　　例

ＰＲＩＮＴ　ＳＵＭ（１０）

⇨１０までの総和５５を表示します。

解　　説

１からｎまでの和、

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots\cdots + (n-1) + n = n(n+1)/2$$

が、この関数の値となります。

**サンプル
プログラム**

```
10 ' SUM <例>
20 INPUT"n=";N
30 PRINT"SUM(n)    =";SUM(N)
40 PRINT"n*(n+1)/2=";N*(N+1)/2
```

２０：変数Ｎに値を入力します。

３０：ＳＵＭ関数で１からＮまでの和を求めます。

４０：公式　Ｎ（Ｎ＋１）／２で１からＮまでの和を求めます。

```
RUN
n=? 100
SUM(n)    = 5050
n*(n+1)/2= 5050
Ok
```

3.1.20 RND

機　能　**0以上1未満の乱数を返します。**

書　式

```
RND [(x)]
```

x：数式。単独の数値定数、数値変数でもかまわない。値は何でもかまいません。

文　例　ＰＲＩＮＴ　ＲＮＤ（１）

⇨0以上１未満の擬似乱数の値を１つ表示します。

解　説

0以上１未満の擬似乱数の値が、この関数の値となります。

乱数の値は単精度型です。

ＲＡＮＤＯＭＩＺＥステートメントを使うと、乱数の初期値を固定することができます。

サンプルプログラム

（例１）

```
10 ' RND (例1)
20 FOR I=1 TO 10
30 PRINT I,RND
40 NEXT
50 END
```

乱数を１0個発生させるプログラムです。このようにＲＮＤ関数は引数なしで使うことができます。

```
RUN
 1            .42706883
 2            .27933753
 3            .55262411
 4            .059644699
 5            .18235326
 6            .54251778
 7            .76047087
 8            .95175505
 9            .29788363
 10           .13059139
Ok
```

（例２）

```
10 ' RND (例2)
20 FOR I=1 TO 10
40 PRINT I,CINT(RND*100),INT(RND*100)
50 NEXT
60 END
```

0～１00までの乱数を発生させるプログラムです。

４0：ＣＩＮＴ（ＲＮＤ＊１00）で0～１00までの乱数が、ＩＮＴ（ＲＮＤ＊１00）で0～９９の乱数が発生します。

```
RUN
 1          28          85
 2          78          14
 3          10          81
 4          58          8
 5          48          1
 6          29          64
 7          37          75
 8          79          0
 9          100         90
 10         63          3
Ok
```

3.2 文字関数

3.2

3.2.1 ASC

機能 文字をコードに変換します。

書式

```
ASC (x$)
```

x $ ：文字列を表わす文字式。単独の文字列、文字変数、文字関数、およびそれらを連結したもの。

文例

PRINT　ASC（"A"）
⇨文字Aのキャラクタコード65を表示します。
PRINT　HEX$（ASC（"愛"））
⇨文字「愛」のシフトJISコード88A4を表示します。

解説

ASCは、1バイトコード文字（半角文字）をキャラクタコード（一般的にはアスキーコードという）、2バイトコード文字（全角文字）をシフトJISコードに変換する関数です。X $の表わす文字列が2字以上のときは、先頭の1字が変換されます。
文字列がヌルストリング（""）のとき、この関数の値は0になります。

参照 A.3「コードの体系」、3.2.2 CHR$

サンプルプログラム

次の例は、入力した文字に対してコードを返すプログラムです。
1バイトコード文字(半角文字)に対してはキャラクタコードを、
2バイトコード文字(半角文字)に対してはシフトJISコードを返します。

なお、ENDを入力するとプログラムが終わります。

```
10 ' ASC (例)
20 INPUT"文字は";A$
30 IF A$="END" THEN END
40 A=ASC(A$)
50 PRINT A$;" のコードは &H";HEX$(A)
60 GOTO20
```

２０：キー入力文字をA$に入れます。

３０：もしA$が"ＥＮＤ"ならばプログラムを終了します。

４０：A$コードを変換します。

５０：結果を１６進数で表示します。

```
RUN
文字は? A
A のコードは &H41
文字は? B
B のコードは &H42
文字は? あ
あ のコードは &H82A0
文字は? 愛
愛 のコードは &H88A4
文字は? END
Ok
```

3.2.2 CHR$

機能　コードを文字に変換します。

書式

```
CHR$ (x1 〔, x2, ……〕)
```

xi：数式。単独の数値定数、数値変数でもかまいません。ただし、その値は0～&HFFFFFの整数。

3.2

文例　PRINT　CHR$ (65)
⇨キャラクタコード65に対応する文字「A」を表示します。
PRINT　CHR$ (&H88A4)
⇨シフトJISコード&H88A4に対応する文字「愛」を表示します。

解説　CHR$は、キャラクタコード（一般的にはアスキーコードという）を1バイトコード文字（半角文字）に、シフトJISコードを2バイトコード文字（全角文字）に変換する関数です。

数式xiの値によって、次のような文字に変換されます。

	xiの値（16進数）	変換される文字
KMODE0の場合	0～FF[1]	半角文字
	100～FFFF[2]	2文字の半角文字
KMODE1の場合	0～FF	半角文字（ただしセミグラフィック文字は表示されない）
	100～FFFF	全角文字（主として漢字）

1）キャラクタコードを示す。
2）シフトJISコードを含む。詳しくはA.3「コードの体系」を参照してください。

参照　A.3「コードの体系」、3.2.1 ASC

サンプルプログラム　例は、入力したコードに対応する文字を返すプログラムです。
入力するコードに16進数で、0～FFに対しては半角文字を、8040以上の値に対しては全角文字を返します。

ENDと入力するとプログラムが終わります。

```
10 ' CHR$ (例)
20 INPUT"コードは &H",X$
30 IF X$="END" THEN END
40 A=VAL("&H"+X$)
50 A$=CHR$(A)
60 PRINT"&H";X$;" に対する文字は ";A$
70 GOTO20
```

２０：キー入力した１６進数文字をＸ$に入れます。

３０：もしＸ$が"ＥＮＤ"ならばプログラムを終了します。

４０：１６進数文字列を数値に変換してＡに代入します。

５０：Ａの値に文字に変換してＡ$に代入します。

６０：結果の文字を表示します。

```
RUN
コードは &H41
&H41 に対する文字は A
コードは &H42
&H42 に対する文字は B
コードは &H82A0
&H82A0 に対する文字は あ
コードは &H88A4
&H88A4 に対する文字は 愛
コードは &HEND
Ok
```

3.2

3.2.3 VAL

機能 文字型の数字を数値に変換します。

書式

```
VAL (x$)
```

x＄：文字列を表わす文字式。単独の文字列、文字変数、文字関数、およびそれらを連結したもの。ただし数字。

3.2

文例 A＝VAL（"２３"）

⇨文字"２３"を数値の２３に変換してAに代入します。

解説 文字列としての数字を数値に変換します。

x＄の表わす数字に、正負の符号（＋、－）、小数点（.）、アンパサンド（&H、&O、&B、&K、&J）、指数部（E～、D～）がついていても、数値に変換することはできます。

x＄の表わす文字列が数字を表わさないとき、この関数の値は０となります。ただし文字列中のスペースは無視します。

参照 3.2.4 STR＄

サンプルプログラム 文字型の数字を数値型の数字に変換するプログラムです。

"END"を入力するとプログラムが終わります。

```
10 ' VAL (例1)
20 INPUT"数字は ";A$
30 IF A$="END" THEN END
40 V=VAL(A$)
50 PRINT"数値は ";V;" です。"
60 GOTO20
```

２０：キー入力した数字をA$に入れます。

３０：A$が"END"ならばプログラムを終了します。

４０：A$を数値に変換してVに代入します。

５０：変換した結果Vを表示します。

```
RUN
数字は ? 10
数値は  10  です。
数字は ? 10000
数値は  10000  です。
数字は ? 100000000
数値は  1E+08  です。
数字は ? &HABCD
数値は -21555  です。
数字は ? 1.02344E-12
数値は  1.02344E-12  です。
数字は ? END
Ok
```

3.2.4 STR$

機　能　**数式の値を文字型の数字に変換します。**

書　式

```
STR$ (x)
```

x：数式。単独の数値定数、数値変数でもかまいません。

文　例　PRINT　STR$（23）
⇨数値23を文字の"23"に変換して表示します。

解　説　数式の値を文字型の数字に変換します。
xが倍精度型実数でも、数字の文字列に変換することができます。
xが正の数のとき、この関数の値は、先頭の1字が空白の文字列になります。

参　照　3.2.3 VAL

サンプルプログラム　次は数値を文字型の数字に変換するプログラムです。0を入力するとプログラムは終わります。

```
10 ' STR$ (例1)
20 INPUT"数値は ";A
30 IF A=0 THEN END
40 A$=STR$(A)
50 PRINT"数字は ";A$;" です。"
60 GOTO20
```

20：キー入力した数字をAに入れます。
30：A＝0ならばプログラムを終わります。
40：Aの値を文字型の数字に変換しA$に代入します。
50：結果を表示します。

```
RUN
数値は ? 10
数字は  10 です。
数値は ? 10000
数字は  10000 です。
数値は ? 1E+08
数字は  1E+08 です。
数値は ? &HABCD
数字は -21555 です。
数値は ? 1.02344E-12
数字は  1.02344E-12 です。
数値は ? 0
Ok
```

3.2.5 HEX$

3.2

機能　**数式の値を16進数表現の文字列に変換します。**

書式

```
HEX$ (x)
```

x：数式。単独の数値定数、数値変数でもかまいません。
　－32768～65535の整数。ただし、－32768～－1は32768～65535と同じ。

省略形　HE.

文例　PRINT　HEX$（255）
⇨255を16進数のFFに変換して表示します。

解説　数式の値を16進数の文字列に変換したものが、この関数の値になります。
この関数は、16進数を表示するプログラムの中で使うと便利です。

参照　3.2.6 OCT$、3.2.7 BIN$

サンプルプログラム　(例1)次の例は、入力した数値を16進数表現の文字列に変換するプログラムです。

```
10 ' HEX$ (例1)
20 INPUT"数値は ";A
30 IF A=0 THEN END
40 H$=HEX$(A)
50 PRINT A;" は16進数表現で ";H$
60 GOTO20
```

20：キー入力した数値をAに入れます。
30：A＝0ならばプログラムを終わります。
40：Aの値を16進数表現の文字列に変換し、H$に代入します。
50：結果を表示します。

```
RUN
数値は ? 1
 1  は16進数表現で 1
数値は ? 10
 10  は16進数表現で A
数値は ? 20
 20  は16進数表現で 14
数値は ? 1000
 1000  は16進数表現で 3E8
数値は ? -1
-1  は16進数表現で FFFF
数値は ? 0
Ok
```

(例2)次は、10進数、2進数、8進数、16進数の対応表を作成するプログラムです。

```
10 ' HEX$ (例2)
15 PRINT"10進数  2進数  8進数  16進数"
20 FOR I=0 TO 16
30 PRINT I;TAB(9);BIN$(I);TAB(16);OCT$(I);TAB(22);HEX$(I)
40 NEXT
```

3.2.6 OCT$

機　能　**数式の値を８進数表現の文字列に変換します。**

書　式

```
OCT$ (x)
```

x：数式。単独の数値定数、数値変数でもかまいません。
－３２７６８～６５５３５の整数。ただし、－３２７６８～－１は３２７６８～６５５３５と同じ。

省略形　ＯＣ.

文　例　ＰＲＩＮＴ　ＯＣＴ＄（２５５）
⇨２５５を８進数３７７に変換し表示します。

解　説　数式の値を８進数の文字列に変換したものが、この関数の値になります。

参　照　3.2.5 ＨＥＸ＄、3.2.7 ＢＩＮ＄

サンプルプログラム

(例１)次のサンプルは、入力した値を８進数表現の文字列に変換するプログラムです。

```
10 ' OCT$ (例1)
20 INPUT"数値は ";A
30 IF A=0 THEN END
40 O$=OCT$(A)
50 PRINT A;" は８進数表現で ";O$
60 GOTO20
```

２０：キー入力した数字をＡに入れます。
３０：Ａ＝０ならば終わります。
４０：Ａの値を８進数表現の文字列に変換してＯ＄に代入します。
５０：結果を表示します。

```
RUN
数値は ? 1
 1  は８進数表現で 1
数値は ? 10
 10  は８進数表現で 12
数値は ? 20
 20  は８進数表現で 24
数値は ? 1000
 1000  は８進数表現で 1750
数値は ? -1
-1  は８進数表現で 177777
数値は ? 0
Ok
```

(例２)次は１０進、１６進、８進、および２進数の対応表を作成するプログラムです。

```
10 ' OCT$ (例2)
15 PRINT"10進数 16進数 8進数 2進数"
20 FOR I=0 TO 16
30 PRINT I;TAB(9);HEX$(I);TAB(16);OCT$(I);TAB(22);BIN$(I)
40 NEXT
```

3.2.7 BIN$

機能　**数式の値を2進数表現の文字列に変換します。**

書式

```
BIN$ (x)
```

x：数式。単独の数値定数、数値変数でもかまいません。
－32768～65535の整数。ただし、－32768～－1は
32768～65535と同じ。

3.2

省略形　BI.

文例　PRINT　BIN$ (255)
⇨255を2進数の11111111に変換して表示します。

解説　数式の値を2進数の文字列に変換したものが、この関数の値になります。
数式xの値は、いったん16ビットの整数型から0と1の文字列に変換され、その上位の0が取り除かれます。たとえば、PRINT　BIN$ (4)は00000100ですが、表示は上位の0を取り除いて100になります。8ビット全部を表示したいときは次のように256をxに加えて、その右8ビット分の表示をするようなプログラムを組んでください。
PRINT　RIGHT$ (BIN$ (X＋256), 8)

参照　3.2.5 HEX$、3.2.6 OCT$

サンプルプログラム　(例1)次の例は、入力した値を2進数表現の文字列に変換するプログラムです。

```
10 ' BIN$ (例1)
20 INPUT"数値は ";A
30 IF A=0 THEN END
40 B$=BIN$(A)
50 PRINT A;" は2進数表現で ";B$
60 GOTO20
```

20：キー入力した値をAに入れます。
30：A＝0ならば終わります。
40：Aの値を2進数表現の文字列に変換してB$に代入します。
50：結果を表示します。

```
RUN
数値は ? 1
 1  は2進数表現で 1
数値は ? 10
 10  は2進数表現で 1010
数値は ? 20
 20  は2進数表現で 10100
数値は ? 1000
 1000  は2進数表現で 1111101000
数値は ? -1
-1  は2進数表現で 1111111111111111
数値は ? 0
Ok
```

(例２)次は１０進、１６進、８進、および２進数の対応表を作成するプログラムです。

```
10 ' BIN$ (例2)
15 PRINT"10進数 16進数 8進数 2進数"
20 FOR I=0 TO 16
30 PRINT I;TAB(9);HEX$(I);TAB(16);OCT$(I);TAB(22);BIN$(I)
40 NEXT
```

3.2

3.2.8 ＬＥＦＴ＄

機　能　**文字列の左側から、指定した数だけの文字を取り出します。**

書　式

```
LEFT$ (x$, n)
```

ｘ＄：文字式。単独の文字列、文字変数、文字関数、およびそれらを連結したもの。

ｎ：文字列から取り出す文字数。０～２５５の整数。

省略形　ＬＥＦ.

文　例　ＰＲＩＮＴ　ＬＥＦＴ＄（ＴＩＭＥ＄，２）

⇨ＴＩＭＥ＄の左側から取り出された２個の文字を表示します。

解　説　ｘ＄の表わす文字列の左側から半角文字でｎ文字（ｎバイト）を取り出します。

ｎは、０～２５５の値をもつ数式で、小数点以下は四捨五入されます。

ｎの値が、文字列の長さＬＥＮ（ｘ＄）より大きい場合は、文字列のすべてが、ｎ＝０のときは、ヌルストリングが、それぞれ関数の値になります。ｎが負のときは、「Illegal function call」のエラーが出ます。

参　照　3.2.9 ＲＩＧＨＴ＄、3.2.10 ＭＩＤ＄

サンプルプログラム

```
10 ' LEFT$ (例)
20 KMODE1
30 INPUT"文字列は ";A$
40 FOR I=0 TO LEN(A$)
50 PRINT LEFT$(A$,I)
60 NEXT
```

３０：入力した文字列をＡ＄に入れます。

４０～６０：Ａ＄の左側からＩ個だけ文字を取り出して画面に表示することをくりかえすＦＯＲ－ＮＥＸＴループです。

```
RUN
文字列は ? Personal Computer

P
Pe
Per
Pers
Perso
Person
Persona
Personal
Personal
Personal C
Personal Co
Personal Com
Personal Comp
Personal Compu
Personal Comput
Personal Compute
Personal Computer
Ok
```

※全角文字の左半分を取り出すことはできません。

3.2.9 RIGHT$

機　能　**文字列の右側から、指定した数だけの文字を取り出します。**

書　式

```
RIGHT$ (x$, n)
```

x＄：文字式。単独の文字列、文字変数、文字関数、およびそれらを連結したもの。

n：文字列から取り出す文字数。0～255の整数。

省略形　RI.

文　例　PRINT　RIGHT$ (TIME$, 2)

⇨TIME＄の右側から取り出された2個の文字を表示します。

解　説

x＄の表わす文字列の右側から半角文字でn文字（nバイト）を取り出します。

nは、0～255の値をもつ数式で、小数点以下は四捨五入されます。

nの値が、文字列の長さLEN（x＄）より大きい場合は、文字列のすべてが、n＝0のときは、ヌルストリングが、それぞれ関数の値になります。nが負のときは、「Illegal function call」のエラーが出ます。

参　照　3.2.8 LEFT＄、3.2.10 MID＄

サンプルプログラム

```
10 ' RIGHT$ (例)
20 KMODE1
30 INPUT"文字列は ";A$
40 FOR I=0 TO LEN(A$)
50 PRINT RIGHT$(A$,I)
60 NEXT
```

30：入力した文字列をA＄に入れます。

40～60：A＄の右側からI個だけ文字を取り出して画面に表示することを繰り返すFOR—NEXTループです。

```
RUN
文字列は ? Personal Compute

r
er
ter
uter
puter
mputer
omputer
Computer
 Computer
l Computer
al Computer
nal Computer
onal Computer
sonal Computer
rsonal Computer
ersonal Computer
Personal Computer
Ok
```

※　全角文字の右半分を取り出そうとするとシフトJISコードの上位バイトに対応する半角文字が表示されます。

3.2.10 MID$

機能　**文字列の中から、一連の文字を指定した数だけ取り出します。**

書式

```
MID$ (x$, m [, n])
```

x＄：文字式。単独の文字列、文字変数、文字関数、およびそれらを連結したもの。２５５文字以内。
m：取り出し始める位置。x＄の表わす文字列の先頭を１としたときの文字の位置。１～２５５の整数。
n：文字列から取り出す文字数。０～２５５の整数。

3.2

省略形　ＭＩ．

文例　ＰＲＩＮＴ　ＭＩＤ＄（ＴＩＭＥ＄，４，２）
⇨ＴＩＭＥ＄の４番目から２個の文字を取り出し表示します。

解説　x＄が表わす文字列の左からm番目の所から半角文字でn文字（nバイト）を取り出します。

nを省略したとき、およびnが文字列の文字数より大きいときは、m番目から右端までのすべての文字列が、この関数の値になります。

mが文字列の文字数より大きいときは、ヌルストリングがこの関数の値になります。

また、この関数を使って、左からm番目の所からn字書き換えることができます。

（例）５０　Ａ＄＝"ＣＡＳ：ＳＴＡＲ．ＢＡＳ"
　　　６０　ＭＩＤ＄（Ａ＄，５，４）＝"ＭＯＯＮ"
　　　　……これを実行すると、Ａ＄はＣＡＳ：ＭＯＯＮ．ＢＡＳという文字列になります。

参照　3.2.8 ＬＥＦＴ＄、3.2.9 ＲＩＧＨＴ＄

サンプルプログラム

（例１）

```
10 ' MID$ (例1-1)
20 KMODE1
30 INPUT"文字列は ";A$
40 FOR I=1 TO LEN(A$)
50 PRINT MID$(A$,I,3)
60 NEXT
```

３０：入力した数字をＡ＄に入れます。
４０～６０：文字列Ａ＄のＩ番目から３文字ずつ取り出して画面に表示することを繰り返すＦＯＲ―ＮＥＸＴループ。

```
RUN
文字列は ? Personal Computer
Per
ers
rso
son
ona
nal
al
l C
 Co
Com
omp
mpu
put
ute
ter
er
r
Ok
```

3.2

（例2）

```
10 ' MID$ (例1-2)
20 KMODE1
30 INPUT"文字列は ";A$
40 FOR I=1 TO LEN(A$)
50 PRINT MID$(A$,I,1)
60 NEXT
```

３０：入力した文字列をＡ＄に入れます。

４０～６０：文字例Ａ＄のＩ番目から１文字ずつ取り出して画面に表示することを繰り返すＦＯＲ－ＮＥＸＴループ。

3.2.11 STRING$

機　　能　文字列を任意の数だけ並べます。

書　　式

〔1〕STRING$ (n, x$)
〔2〕STRING$ (n, x)

n：並べる回数。数式。単独の数値定数、数値変数でもかまいません。値は0〜255の整数。
x$：並べる文字式。単独の文字列、文字変数、文字関数、およびそれらを連結したもの。
x：並べる数式。単独の数値定数、数値変数でもかまいません。値は0〜255の整数でキャラクタコードを表わします。

省 略 形　STRIN.

文　　例　PRINT STRING$ (3, "ABC")
⇨文字ABCが3つ並んで「ABCABCABC」と表示されます。

解　　説　x$の表わす文字列、またはキャラクターコードxの表わす文字をn個並べて用意します。

〈注　　意〉　ただし文字式の場合、その文字式のバイト数と並べる数を指すnの積が255を越えると、「Illegal function call」のエラーが出ます。

サンプルプログラム

STRING$を使って、棒グラフを描くプログラムを作ってみましょう。

```
10 ' STRING$ (例1-1)
20 FOR X=0 TO PAI(1) STEP PAI(1)/20
30 S=SIN(X)
40 PRINTUSING"#.#### ";X;S;
50 PRINT STRING$(20*S,"*")
60 NEXT
```

20〜60：X=0からπまでのSINグラフを描きます。
30：SIN (X) の値をSに代入します。
40：XとSの値を表示します。
50：−1〜1の範囲のSの値を20倍して、その数の"＊"を用意します。

```
RUN
0.0000 0.0000
0.1571 0.1564 ***
0.3142 0.3090 ******
0.4712 0.4540 *********
0.6283 0.5878 ************
0.7854 0.7071 **************
0.9425 0.8090 ****************
1.0996 0.8910 ******************
1.2566 0.9511 *******************
1.4137 0.9877 ********************
1.5708 1.0000 ********************
1.7279 0.9877 ********************
1.8850 0.9511 *******************
2.0420 0.8910 ******************
2.1991 0.8090 ****************
2.3562 0.7071 **************
2.5133 0.5878 ************
2.6704 0.4540 *********
2.8274 0.3090 ******
2.9845 0.1564 ***
3.1416 0.0000
Ok
```

3.2

3.2.12 SPACE$

機能 空白からなる文字列を返します。

書式

```
SPACE$ (n)
```

n：数式。空白の数を表わす0～255の整数。単独の数値定数、数値変数でもかまいません。

3.2

省略形 SPA.

文例 A$=SPACE$(32)
⇨32文字分の空白をA$として用意します。

解説 n個の空白からなる文字列が、この関数の値になります。

参照 2.3.6 PRINTの項─SPC

サンプルプログラム

```
10 ' SPACE$ (例)
20 S$=SPACE$(10)
30 A$="A"
40 PRINT S$;A$
```

20：10個の空白からなる文字列をS$に入れます。

```
RUN
          A
Ok
```

3.2.13　ＬＥＮ

機能

文字列の長さ（文字数）を返します。

書式

```
LEN (x$)
```

x＄：文字式。単独の文字列、文字変数，文字関数、およびそれらを連結したもの。０～２５５の長さの文字列を表わす。

3.2

文例

PRINT　LEN（A$）

⇨文字列A＄の長さを表示します。

解説

文字列に含まれる文字（画面に表示されないコントロールコードや空白も含む）をすべて数え、その長さがこの関数の値となります。

２バイトコード文字（全角文字）は、２文字として数えられます。２バイトコード文字を１文字として数えたいときは、3.4.4　KLENを使用してください。

x＄の表わす文字列が、ヌルストリングのとき、関数の値は０となります。

参照

3.4.4 KLEN

サンプルプログラム

```
10 ' LEN (例)
20 INPUT"文字列は ";A$
30 L=LEN(A$)
40 PRINT"長さは";L;"です。"
```

２０：入力した文字列をA＄に入れます。

３０：A＄の文字数を数えてLに代入します。

４０：結果を表示します。

```
RUN
文字列は ? ABCDEFGHIJKLMNOPQRSTUVWXYZ
長さは 26 です。
Ok
```

3.2.14　INSTR

機　能　**文字列の中から任意の文字列を探し、その位置を返します。**

書　式

〔1〕 INSTR (x$, y$)
〔2〕 INSTR (n, x$, y$)

x $：文字式。単独の文字列、文字変数、文字関数、およびそれらを連結したもの。もとになる文字列を表わす。
y $：文字式。文字列、単独の文字列、文字変数、文字関数、およびそれらを連結したもの。探したい文字列を表わす。
n：探し始める文字の位置。1～255の整数。

省略形　INS.

文　例　PRINT　INSTR (A$, "X")
⇨文字列A$の中に文字"X"があるかどうか探し、あれば何番目にあるか表示します。

解　説　x$で表わされる文字列を「文字列x」、y$で表わされる文字列を「文字列y」とすると、文字列xの中に文字列yが含まれているかどうか探し、含まれている場合は、その先頭文字の位置（左から何文字目か）を関数の値とします。含まれていない場合は、0が関数の値となります。ただし、文字列yがヌルストリング("　")の場合、文字列xにかかわらず1が関数の値になります。
2バイトコード文字（全角文字）は、2文字として数えられます。
なお、探し始める文字の位置nを指定することができます。

参　照　3.4.6 KPOS

サンプルプログラム

```
10 ' INSTR <例>
20 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30 PRINT INSTR(A$,"X")
```

20：A$にアルファベット26文字の文字列を代入します。
30：A$の中から"X"を探し、その位置を返します。

```
RUN
 24
Ok
```

3.2.15 MIRROR$

機　能

文字列の２進数コードを転置した文字列を返します。

書　式

```
MIRROR$ (x$)
```

x $：文字式。単独の文字列、文字変数、文字関数、およびそれらを連結したもの。

3.2

省略形

MIR.

文　例

PRINT　MIRROR$ ("K")
⇨文字"メ"が表示されます。

解　説

文字列の２進数コードを左右ひっくり返したコードに対応する文字列が、この関数の値になります。

（例１）MIRROR$ ("K") ……この値は、文字の"メ"（引用符は含まない）となります。

なぜなら、Kのキャラクタコードは&H4Bで、これを２進数コードになおすと、

0 1 0 0 1 0 1 1

これをひっくり返すと、

1 1 0 1 0 0 1 0

となり、これはキャラクタコードの&HD2で、文字の"メ"に対応しています。

２進数コードの転置は、１バイト（８ビット）ごとに行なわれます。

（例２）２進数コード010010110100110001001101は、

0 1 0 0 1 0 1 1　0 1 0 0 1 1 0 0　0 1 0 0 1 1 0 1
⇩　⇩　⇩
1 1 0 1 0 0 1 0　0 0 1 1 0 0 1 0　1 0 1 1 0 0 1 0

となります。

サンプルプログラム

```
10 ' MIRROR$ (例1)
20 A$=CHR$(&H4B)
30 PRINT A$,MIRROR$(A$)
```

20：キャラクタコード&H4Bに対する文字KがA$に代入されます。
30：&H4Bを転置したコードは&HD2なのでMIRROR$ (A$) はカタカナのメになります。

```
RUN
K             ﾒ
Ok
```

3.2.16 HEXCHR$

機能

16進数の文字列をキャラクタコード列とみなして、対応する文字列を返します。

書式

```
HEXCHR$ (x$)
```

x$：文字式。単独の文字列、文字変数、文字関数、およびそれらを連結したもの。16進数表現の文字列となっていなければなりません。

3.2

省略形

HEXC.

文例

PRINT　HEXCHR$ ("53756E")

⇨16進数53、75、6Eに対応する文字Sとuとnを表示します。

解説

x$の表わす16進数表現の文字列を、先頭から2文字（2バイト）ずつ区切って、16進数2けたのキャラクタコードとみなし、対応する文字に変換して文字列を作ります。この文字列が、この関数の値となります。

16進数表現の文字列の中に空白があるときは、文字列の区切れとして無視します。

HEXCHR$は、グラフィックに関するステートメントの中で、比較的大きなビットパターンを扱うのに便利です。

（例）PAINT、LINEステートメントのタイリングパターン

PATTERNステートメントのドットパターン

DEFCHR$ステートメントのドットパターン

参照

2.8.5 LINE、2.8.9 PAINT、2.8.12 PATTERN

2.7.9 DEF　CHR$

サンプルプログラム

```
10  ' HEXCHR$ (例)
20 S$=""
30 FOR I=&H41 TO &H5A
40 S$=S$+HEX$(I)+"  "
50 NEXT
60 T$=HEXCHR$(S$)
70 PRINT S$
80 PRINT T$
```

20～50：41から5Aまでの16進表現の文字列を作ってS$に入れます。

60：16進文字列S$に対応する文字列をT$に代入します。

70、80：S$とT$を表示します。

```
RUN
41  42  43  44  45  46  47  48  49  4A
4B  4C  4D  4E  4F  50  51  52  53  54
55  56  57  58  59  5A
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ok
```

3.2.17 MKI$／MKS$/MKD$

機能

数式をそのまま内部表現（バイナリィコード）に対応した文字列に変換します。

書式

〔1〕MKI$（$x_1$）
〔2〕MKS$（$x_2$）
〔3〕MKD$（$x_3$）

x_1：整数型の数式。単独の整数、数値変数でも使えます。
x_2：単精度型の数式。単独の単精度型実数、数値変数でも使えます。
x_3：倍精度型の数式。単独の倍精度型実数、数値変数でも使えます。

省略形

〔1〕MKI.
〔2〕ありません。
〔3〕ありません。

文例

LSET　A$＝MKI$（X％）
⇨整数値をもつX％を2バイトの文字列に変換してA$に左詰めで代入します。

解説

MKI$は、2バイトの整数の値を2バイト（＝2文字）の文字列に、
MKS$は、5バイトの単精度型実数の値を5バイト（＝5文字）の文字列に、
MKD$は、8バイトの倍精度型実数の値を8バイト（＝8文字）の文字列に、
それぞれ変換します。

LSETやRSETでファイルバッファにデータを書き込むとき、そのデータは文字型である必要があります。このため、数値型データを書き込むときは、MKI$、MKS$、MKD$を使って、文字型データに変換するようにします。

逆に、GETでファイルバッファからデータを読み込むときには、CVI、CVS、CVD関数を使って、データを文字型から数値型に変換します。

参照

『ユーザーズマニュアル』の「プログラム、データーファイルの保存と再生」
3.2.18 CVI／CVS／CVD

サンプル
プログラム

ＭＫＩ＄、ＭＫＳ＄、ＭＫＤ＄は、数値型データをランダムアクセスファイルに書き込む途中の過程で使います。

すなわち、数値型データはそのままファイルバッファに書き込めないのでＭＫＩ＄、ＭＫＳ＄、ＭＫＤ＄を使って文字型に変換します。

```
10 ' MKI$/MKS$/MKD$ (例)
20 INIT:WIDTH40,25
30 MAXFILES1
40 N=0
50 INPUT"ﾌｧｲﾙ名";NM$
60 OPEN"R",#1,"1:"+NM$
70 FIELD#1,5 AS X$
80 LABEL"入力":N=N+1
90 INPUT"ﾃﾞｰﾀ";X
100 IF X=-1 THEN "ｸﾛｰｽﾞ"
110 LSET X$=MKS$(X)
120 PUT#1,N
130 GOTO"入力"
140 LABEL"ｸﾛｰｽﾞ"
150 CLOSE#1
160 END
```

２０：画面の初期化を行ないます。

３０：オープンできるファイルの数を１つにします。

４０：入力データのカウンターＮの初期化をします。

５０：作成したいファイルの名前を入力します。

６０：ディスクのドライブ１をファイル番号１のランダムアクセスファイルとしてオープンします。

７０：ファイルバッファ内に変数Ｘ＄を５バイト割りつけます。
５バイトとしたのは単精度型の数値データを扱うためです。

８０～１３０：データの入力ループ。

９０：数値型のデータを入力してＸに入れます。

１００：Ｘが－１のときは"クローズ"へジャンプしてファイルをクローズします。

１１０：Ｘの値を５バイトの文字型データに変換してファイルバッファに書き込みます。

１２０：データをランダムアクセスファイルのＮ番レコードに転送します。

１３０：８０行へ戻ります。

3.2.18 ＣＶＩ／ＣＶＳ／ＣＶＤ

機能 **文字型の値を数値型の値とみなします。**

書式

```
〔1〕CVI (x1$)
〔2〕CVS (x2$)
〔3〕CVD (x3$)
```

x_1＄：2バイト（2文字）の文字列。
x_2＄：5バイト（5文字）の文字列。
x_3＄：8バイト（8文字）の文字列。

文例 Ｘ％＝ＣＶＩ（Ａ＄）
⇨2バイトの文字を整数に変換してＸ％に代入します。

解説 ＣＶＩは、2バイト（＝2文字）の文字列を整数値に、
ＣＶＳは、5バイト（＝5文字）の文字列を単精度型実数値に、
ＣＶＤは、8バイト（＝8文字）の文字列を倍精度型実数値に、
それぞれ変換します。

ＧＥＴでファイルバッファからデータを読み込むとき、そのデータは文字型になっています。このため、それを数値として読み込むときには、ＣＶＩ、ＣＶＳ、ＣＶＤ関数を使って、文字型から数値型に変換する必要があります。

逆に、ＬＳＥＴやＲＳＥＴでファイルバッファにデータを書き込むとき、そのデータは文字型である必要があります。このため、数値型データを書き込むときには、ＭＫＩ＄、ＭＫＳ＄、ＭＫＤ＄を使って、文字型データに変換します。

ＣＶＩ、ＣＶＳ、ＣＶＤ関数の使い方について、詳しくは『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」を参照してください。

参照 『ユーザーズマニュアル』の「プログラム、データファイルの保存と再生」
3.2.17 ＭＫＩ＄、ＭＫＳ＄、ＭＫＤ＄

サンプル
プログラム

ＣＶＩ、ＣＶＳ、ＣＶＤは、ランダムアクセスファイルから読み出した文字型データを数値型に変換するときに使います。

```
10 ' CVI/CVS/CVD (例)
20 INIT:WIDTH40,25
30 MAXFILES1
40 N=0
50 INPUT"ファイル名";NM$
60 OPEN"R",#1,"1:"+NM$
70 FIELD#1,5 AS X$
80 LABEL"読み出し":N=N+1
90 IF LOF(1)<N THEN "クローズ"
100 GET#1,N
110 X=CVS(X$)
120 PRINT X,
130 GOTO"読み出し"
140 LABEL"クローズ"
150 CLOSE#1
160 END
```

２０：画面の初期化を行ないます。

３０：オープン可能なファイルの数を１つにします。

４０：レコード番号カウンターＮを初期化します。

５０：アクセスしたいファイルのファイル名を入力します。

６０：ディスクのドライブ１のファイルをファイル番号１のランダムアクセスファイルとしてオープンします。

７０：ファイルバッファに変数Ｘ＄の領域を５バイト割りつけます。

８０～１３０：データの読み出しループ。

９０：レコード番号がファイルのサイズより大きくなったら"クローズ"へジャンプしてファイルをクローズします。

１００：Ｎ番レコードから文字型データを読み出します。

１１０：文字型データを数値型に変換してＸに入れます。

１３０：８０行へ戻ります。

3.3 入出力用文字関数

3.3

3.3.1 MEM$

機　能　指定したメモリから任意の数だけ文字を取り出します。

書　式

```
MEM$ (a, n)
```

a：アドレス、0〜&HFFFF。
n：書き込む文字の数。0〜255の整数。

文　例　D$=MEM$（&HC000，6）
⇨メモリ中のアドレス&HC000から始まるエリアから6個の文字を取り出してD$に代入します。

解　説　メインメモリ内のアドレスaから始まるnバイトのデータを文字列として取り出します。

参　照　2.17.1 MEM$ステートメント

サンプルプログラム

次は、MEM$文で、メモリのE000番地以降に26バイトのアルファベットを書き込み、MEM$関数でE000番地から5バイトとE015番地から5バイトを取り出して画面に表示するプログラムです。

```
10 ' MEM$ (例)
20 CLEAR &HE000
30 MEM$(&HE000,26)="ABCDEFGHIJKLMNOPQRST
UVWXYZ"
40 A$=MEM$(&HE000,5)
50 B$=MEM$(&HE015,5)
60 PRINT A$
70 PRINT B$
```

30：メモリのE000番地以降に26文字のアルファベットを書き込みます。
40：E000番地から取り出してA$に入れます。
50：E015番地から5文字取り出してB$に入れます。
60、70：A$とB$を画面に表示します。

```
RUN
ABCDE
VWXYZ
Ok
```

3.3.2 SCRN$

機　　能　テキスト画面上の文字列を返します。

書　　式

```
SCRN$ (x, y, n)
```

x，y：カーソルの座標。
　x：横座標。　ＷＩＤＴＨ　８０のとき、０～７９の整数。
　　　　　　　ＷＩＤＴＨ　４０のとき、０～３９の整数。
　y：たて座標。テキスト１０行のとき、０～９の整数。
　　　　　　　テキスト１２行のとき、０～１１の整数。
　　　　　　　テキスト２０行のとき、０～１９の整数。
　　　　　　　テキスト２５行のとき、０～２４の整数。
n：文字の数。０～２５５の整数。

文　　例　Ａ＄＝ＳＣＲＮ＄（０，０，１０）

⇨テキスト画面上の左上隅（ホーム位置）から、表示されている１０個の文字をＡ＄に代入します。

解　　説　テキスト画面上の指定位置（x，y）から、半角文字でn個分の文字を取り出します。

n＝０のとき、この関数の値はヌルストリングになります。

参　　照　3.3.3 ＣＨＡＲＡＣＴＥＲ＄、2.7.10 ＰＯＫＥ＠、3.6.3 ＰＥＥＫ＠

サンプルプログラム

```
10 ' SCRN$ <例>
20 INIT:WIDTH40,25:CLS
30 LOCATE0,0:PRINT"ABCDEFGHIJKLMNOPQRSTU
VWXYZ"
40 FOR I=1 TO 10
50 A$=SCRN$(I,0,5)
60 LOCATE0,I:PRINT A$
70 NEXT

RUN


ABCDEFGHIJKLMNOPQRSTUVWXYZ
BCDEF
CDEFG
DEFGH
EFGHI
FGHIJ
GHIJK
HIJKL
IJKLM
JKLMN
KLMNO
Ok
```

3.3.3 CHARACTER$

機能　テキスト画面上の文字を返します。

書式

```
CHARACTER$ (x, y)
```

x，y：カーソルの座標。
　x：横座標。　ＷＩＤＴＨ　８０のとき、０～７９の整数。
　　　　　　　ＷＩＤＴＨ　４０のとき、０～３９の整数。
　y：たて座標。テキスト１０行のとき、０～９の整数。
　　　　　　　テキスト１２行のとき、０～１１の整数。
　　　　　　　テキスト２０行のとき、０～１９の整数。
　　　　　　　テキスト２５行のとき、０～２４の整数。

省略形　ＣＨＡＲ.

文例　Ａ＄＝ＣＨＡＲＡＣＴＥＲ＄（０，０）
⇨テキスト画面のホーム位置にある文字をＡ＄に代入します。

解説　テキスト画面上の指定位置（x，y）から、１文字を取り出します。
　これは、ＳＣＲＮ＄において、ＳＣＲＮ＄（x，y，１）と指定した場合と同じです。

参照　3.3.2 ＳＣＲＮ＄、2.7.10 ＰＯＫＥ＠、3.6.3 ＰＥＥＫ＠

サンプルプログラム

```
10 ' CHARACTER$ (例)
20 INIT:WIDTH40,25:CLS
30 LOCATE0,0:PRINT"ABCDEFGHIJKLMNOPQRSTU
VWXYZ"
40 FOR I=1 TO 10
50 A$=CHARACTER$(I,0)
60 LOCATE0,I:PRINT A$
70 NEXT


RUN

ABCDEFGHIJKLMNOPQRSTUVWXYZ
B
C
D
E
F
G
H
I
J
K
Ok
```

3.3.4 CGPAT$

機　能

キャラクタゼネレータに定義されている文字のドットパターンをビットデータとして返します。

書　式

```
CGPAT$ (n)
```

n：キャラクタコードを表わす数式。単独の定数、数値変数でも使えます。

省略形

CGP.

3.3

文　例

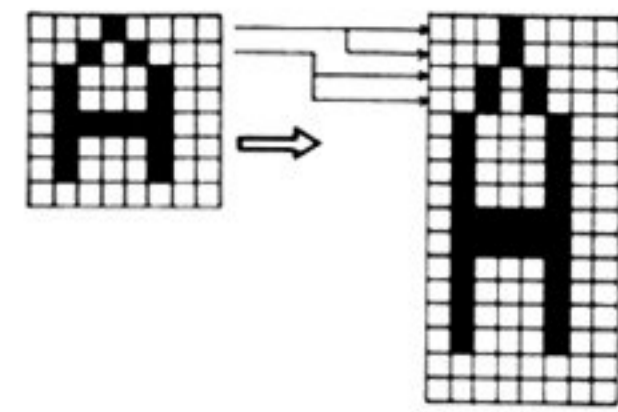
PRINT#0　CGPAT$（65）
⇨キャラクタコード65に対応する32バイトの文字列を表示します。

解　説

キャラクタゼネレータに定義されている文字のドットパターンをビットデータとして返します。
コードnの値によって、次のような文字列が返されます。

nの値（16進数）	対　象	バイト数
0～FF	ROMCG + RAMCG（B. R. G）	8×4＝32
100～1FF	ROMCG	8
200～2FF	RAMCG（B. R. G）	8×3＝24
300～3FF	$ROMCG_1$	16
400～4FF	$RAMCG_1$（B. R. G）	16×3＝48
500～5FF	$ROMCG_2$	16
600～6FFの偶数値	$RAMCG_2$（B. R. G）	16×3＝48
8140～EA9E（シフトJISコード）	漢字	32
EB9F～EBDE（シフトJISコード）	外字（B. R. G）	32×3＝96

※ROMCG：ここに定義されているキャラクタは、8×8のドットパターンで、縦1×横8の部分が8けたのビットパターン（8ビットコード）に対応する文字を表わし、これが縦に8ライン分あるので8バイトの文字列を形成します。（PATTERN参照）

ＲＯＭＣＧ１：ここに定義されているキャラクタは、１６×８のドットパターンで８×８ドットのキャラクタを縦に２倍に拡大したもの（１×８のドットパターンを２回用いている）なので、１６バイトの文字列を形成します。(図参照)

ＲＯＭＣＧ２：ここに定義されているキャラクタは、ＣＧ内の１６×８のドットパターンで、１６バイトの文字列を形成します。

注）　ＲＯＭＣＧには、８×８と１６×８のドットパターンのキャラクタが入っています。

ＲＡＭＣＧ：８×８のドットパターンで、青Ｂ、赤Ｒ、緑Ｇの３原色が８バイトずつ集まって（８＋８＋８）＝２４バイトの文字列を形成します。(ＤＥＦ ＣＨＲ＄参照)

ＲＡＭＣＧ１：１文字８×８のドットパターンを１６×８のドットパターンのキャラクタとして表現するので、ＲＡＭＣＧのパターンをちょうど２倍にした４８バイトの文字列を形成します。Ｂ．Ｒ．Ｇは１６バイトずつで（１６＋１６＋１６）＝４８バイトで、ＲＯＭＣＧ１の図と同じ様になっています。

ＲＡＭＣＧ２：１６×８のドットパターンでＲＡＭＣＧ１とは異なり、８×８ドットパターンのキャラクタを２キャラクタ分使って表現するので、４８バイトの文字列を形成します。Ｂ．Ｒ．Ｇは１６バイトずつ（１６＋１６＋１６）＝４８バイトです。

外字：『ユーザーズマニュアル』の「ユーザー定義文字の使いかた」参照。

参　照　　2.8.11 ＰＡＴＴＥＲＮ、2.7.9 ＤＥＦ　ＣＨＲ＄

サンプルプログラム

```
10 ' CGPAT$ (例)
20 INIT:WIDTH40,25,0:KMODE 1
30 A$=HEXCHR$("103C103E55593200")          "あ"のパターンの設定。
40 DEFCHR$(65)=A$+A$+A$
50 C$=HEXCHR$("0000001C2241410106080800008080000")   "?"のパターンの設定。
60 DEFCHR$(&H100+66)=C$
70 B$="007F0202027F4242 42427F0202027F00 00FE40404   "亜"の外字パターンの
0FE4242 4242FE404040FE00"                            設定。
80 DEFCHR$(&J7622)=HEXCHR$(B$)
90 POSITION100,50
100 PATTERN-32,CGPAT$(65)            ←①
110 POSITION110,50
120 PATTERN-24,CGPAT$(&H100+65)      ←②
130 POSITION120,50
140 PATTERN-32,CGPAT$(&H200+65)      ←③
150 POSITION130,50
160 PATTERN-32,CGPAT$(&H300+65)      ←④
170 POSITION140,50
180 PATTERN-48,CGPAT$(&H400+65)      ←⑤
190 POSITION155,50
200 PATTERN-32,CGPAT$(&H500+65)      ←⑥
210 POSITION170,50
220 PATTERN-48,CGPAT$(&H600+66)      ←⑦
230 POSITION190,50
240 PATTERN-16,CGPAT$(&J7622)        ←⑧
250 END
RUN
```

① ② ③ ④ ⑤ ⑥ ⑦ ⑧

A はROMCGから読み込まれたドットパターンです。

あ、?、亜、はRAMCGから読み込まれたドットパターンです。

3.3.5　INKEY$

機能　**キーボードから入力した1文字を返します。**

書式

```
〔1〕INKEY$
〔2〕INKEY$ (n)
```

n：入力のタイプ。0～2の整数。

省略形　INK.

文例　IF　INKEY$="A"　THEN　1000

⇨キーボードから入力された文字が大文字のAならば行番号1000へジャンプします。

解説　キーボードから入力された1文字が、この関数の値になります。

nの値によって、次の4つのタイプに分けられます。

(1)(n)を省略した場合

キーを押すと、その文字が返り、キーを離すと、ヌルストリングが返ります。

(2) n＝0の場合

キーを押すと、その文字が間断なく返り、キーを離すと、ヌルストリングが返ります

(3) n＝1の場合

カーソルをブリンキングし、1文字の入力があるまで待ちます。1文字入力のINPUT$と考えることができます。

(4) n＝2の場合

キーボードからの入力状態を示すサブCPUからの1バイトのデータが入ります。入力状態を示す1バイトのビット構成は、次のようになっています。

ビット7　6　5　4　3　2　1　0

T	P	R	G	L	K	S	C

1バイトデータ

ビット	得　ら　れ　る　情　報
T	テンキー部からの入力※……………………… 0：有／1：無
P	キーの入力……………………………………… 0：有／1：無
R	リピート機能の有無……………………………… 0：有／1：無
G	GRAPH キーの入力……………………… 0：有／1：無
L	CAPS LOCK キーの入力………………………… 0：有／1：無
K	カナ キーの入力……………………… 0：有／1：無
S	SHIFT キーの入力……………………… 0：有／1：無
C	CTRL キーの入力……………………… 0：有／1：無

※ XFER ROLL UP ROLL DOWN HELP COPY キーも含む

1バイトのデータを8けたの2進数に変換するためのプログラムを次に示します。

(例)
```
10 A$=INKEY$(2)
20 B$=BIN$(ASC(A$))
30 C$=RIGHT$("00000000"+B$,8)
40 PRINT C$
50 GOTO 10
```

参　照　3.3.6 INPUT$

サンプルプログラム

次の(例1)、(例2)、(例3)をそれぞれ実行し、実際にキーボードのキーを押しながら、どのように値が入力されるのか調べてみてください。

(例1)
```
10 ' INKEY$ (例)
20 K$=INKEY$
30 PRINT ASC(K$);
40 GOTO20
```

(例2)
```
10 ' INKEY$ (例2)
20 K$=INKEY$(0)
30 PRINT ASC(K$);
40 GOTO20
```

(例3)
```
10 ' INKEY$ (例3)
20 K$=INKEY$(1)
30 PRINT ASC(K$);
40 GOTO20
```

INKEY$(1)は、INPUT$(1)と同じ動作をします。

```
10 ' INPUT$
20 K$=INPUT$(1)
30 PRINT ASC(K$);
40 GOTO10
```

（例４）

```
10 ' INKEY$ (例4)
20 K$=INKEY$(2)
30 B$=BIN$(ASC(K$))
40 R$=RIGHT$("00000000"+B$,8)
50 PRINT R$
60 GOTO20
```

3.3

3.3.6 INPUT$

機　能

キーボードあるいはファイルから文字列を読み込みます。

書　式

```
〔1〕 INPUT$ (m)
〔2〕 INPUT$ (m, #n)
```

m：読み込む文字の数。0～255の整数。
n：OPENで指定したファイル番号。1～15の整数。

省略形

I.$

3.3

解　説

キーボードからm字だけ文字を読み込みます。

INPUTステートメントとは違い、入力した文字列のおしまいに⏎キーを押す必要がなく、画面上に「?」、入力した文字が表示されません。また、BREAKキーのコントロールコード（&H03）以外のすべてのキャラクタコードを読み込むことができます。

（例1） 100　A$=INPUT$（1）…………キーボードから1文字だけ文字を読込みます。

ファイル番号nを書くと、そのファイルからn字読み込むことができます。

そのとき、前もってシーケンシャル入力モードでファイルをオープンしておかなければなりません。

（例2） 100　A$=INPUT$（5, #1）…………ファイル番号1のファイルから5文字読み込みます。

参　照

3.3.5 INKEY$、2.4.3 OPEN

サンプルプログラム

```
10 ' INPUT$ (例1)
20 K$=INPUT$(1)
30 PRINT ASC(K$);
40 GOTO20
```

3.4 日本語文字関数

3.4.1 AKCNV$

機能

文字列中の1バイトコード文字を2バイトコード文字に変換します。

書式

```
AKCNV$ (x$)
```

x $：文字式。"文字列"、文字変数、文字関数、およびそれらを連結したもの。

省略形

AK.

文例

PRINT AKCNV$ ("A")

⇨半角（1バイトコード文字）Aを全角（2バイトコード文字）**Ａ** に変換し表示します。

解説

x $で表わされる文字列の中の1バイトコード文字（半角文字）を、2バイトコード文字（全角文字）に変換します。

2バイトコード文字に変換された文字は、横2倍の大きさに拡大されます。

なお、文字列中の2バイトコード文字はそのままで変換されません。

参照

3.4.3 KACNV$

サンプルプログラム

```
10 ' AKCNV$ (例)
20 A$="ABCabc123ｱｲｳあいうぁぃぅ愛"
30 PRINT AKCNV$(A$)

RUN
ＡＢＣａｂｃ１２３アイウあいうぁぃぅ愛
Ok
```

全角文字の「あいうぁぃぅ愛」は変化しません。

3.4.2 ＪＩＳ＄

機能

全角文字（２バイトコード文字）のＪＩＳ漢字コードを返します。

書式

```
JIS$（x$）
```

ｘ＄：全角文字（２バイトコード文字）の文字式。"文字列"、文字変数、文字関数、およびそれらを連結したもの。

省略形

Ｊ.

文例

ＰＲＩＮＴ　ＪＩＳ＄（"愛"）

⇨　のＪＩＳ漢字コード「３０２６」が表示されます。

解説

ｘ＄が表わされる文字列の最初の２バイトを１６進数表現４けたの文字列に変換します。このとき、この４けたの１６進数をＪＩＳ漢字コードといい、頭に「&Ｊ」をつけてＪＩＳ１６進定数として扱うことができます。

（例）　ＪＩＳ＄（"漢"）…………ＪＩＳ漢字コードの３４４１が返されます。

文字列の最初の２バイトが２バイトコード文字でないときは、「Ｉｌｌｅｇａｌ　ｆｕｎｃｔｉｏｎ　ｃａｌｌ」のエラーが出ます。

参照

3.4.5 ＫＴＮ＄、付録Ａ.3.3「非漢字およびＪＩＳ第１水準漢字一覧表」

サンプルプログラム

```
10 ' JIS$ (例)
20 INPUT"日本語文字は ";X$
30 A$=JIS$(X$)
40 PRINT X$;" のJIS漢字コードは ";A$


RUN
日本語文字は ? 愛
愛 のJIS漢字コードは 3026
Ok
```

3.4.3 KACNV$

機能

文字列中の全角文字（2バイトコード文字）を半角文字（1バイトコード文字）に変換します。

書式

```
KACNV$ (x$)
```

x＄：文字式。"文字列"、文字変数、文字関数、およびそれらを連結したもの。

省略形

KAC.

文例

PRINT KACNV$ ("Ａさん")
⇨「Ａさん」を半角文字「Aｻﾝ」に変換して表示します。

解説

x＄で表わされる文字列中の全角文字（2バイトコード文字）を、半角文字（1バイトコード文字）に変換します。

ただし、文字列中の1バイトコード文字（半角文字）はそのままで変換されません。

なお、変換できない文字はそのまま表示されます。

参照

3.4.1 AKCNV$

サンプルプログラム

```
10 ' KACNV$ (例)
20 A$="ABCabc123ｱｲｳＡＢＣあいうぁぃぅ愛"
30 PRINT KACNV$(A$)

RUN
ABCabc123ｱｲｳABCｱｲｳｧｨｩ愛
Ok
```

半角文字「ＡＢＣabc１２３アイウ」および漢字は変化しません。また全角文字のひらがなは半角のカタカナに変換されます。

3.4.4 KLEN

機　能　全角文字（2バイトコード文字）を含む文字列の文字数を返します。

書　式

```
KLEN (x$)
```

x $：文字式。"文字列"、文字変数、文字関数、およびそれらを連結したもの。

省略形　KLE.

文　例　PRINT KLEN ("　21世紀へ向かって　")

⇨「　21世紀へ向かって　」の文字数9が表示されます。

3.4

解　説　1バイトコード文字も2バイトコード文字も1文字として数えて、x $で表わされる文字列の文字数を返します。

このとき、1バイトコード文字、2バイトコード文字の空白も1文字に数えます。

参　照　3.2.13 LEN

サンプルプログラム

```
10 ' KLEN (例)
20 INPUT"文字列は ";X$
30 L=KLEN(X$)
40 PRINT X$;" の文字数は ";L


RUN
文字列は ? 21世紀へ向かって
21世紀へ向かって の文字数は  9
Ok
```

3.4.5 ＫＴＮ＄

機　能　**全角文字（2バイトコード文字）の区点コードを返します。**

書　式

```
KTN$ (x$)
```

ｘ＄：文字式。"文字列"、文字変数、文字関数、およびそれらを連結したもの。

省略形　ＫＴ.

文　例　ＰＲＩＮＴ　ＫＴＮ＄（"愛"）

⇨愛の区点コード「１６０６」が表示されます。

解　説　ｘ＄で表わされる文字列の最初の１字を区点コードに変換します。

文字列の最初の１字が２バイトコード文字でないときは、「Ｉｌｌｅｇａｌ　ｆｕｎｃｔｉｏｎ　ｃａｌｌ」のエラーが出ます。

参　照　3.4.2　ＪＩＳ＄、付録Ａ.3.3「非漢字およびＪＩＳ第１水準漢字一覧表」

サンプルプログラム

```
10 ' KTN$ (例)
20 INPUT"日本語文字は ";X$
30 A$=KTN$(X$)
40 PRINT X$;" の区点コード は ";A$


RUN
日本語文字は ? 愛
愛 の区点コード は 1606
Ok
```

3.4

3.4.6 KPOS

機　能　文字列の先頭からの位置を返します。

書　式

```
KPOS (x$, n)
```

x $ ：文字式。単独の文字列、文字変数、文字関数、およびそれらを連結したもの。

n ：数式。単独の定数、数値定数、数値関数でもかまいません。
0～255の整数。

省略形　KP.

文　例　PRINT KPOS ("21世紀へ向かって", 6)

⇨「21世紀へ向かって」の6番目の文字「向」が9バイト目にあるので、9が表示されます。

解　説

x $で表わされる文字列のn番目の文字の位置をバイト数で与えます。

数式nは、文字列の文字を先頭から数えたときの値で、1バイトコード文字も2バイトコード文字も1文字とみなします。

この関数は、文字列中から任意の文字列を取り出すときに使うと便利です。

参　照　3.2.14 INSTR

サンプルプログラム

```
10 ' KPOS (例)
20 INPUT"文字列は ";X$
30 A$=MID$(X$,KPOS(X$,3),4)
40 PRINT A$
```

```
RUN
文字列は ? 21世紀へ向かって
世紀
Ok
```

3.5　ファイル用関数

3.5.1　LOC

機　能　ファイルの中の現在位置を返します。

書　式

```
LOC (n)
```

ｎ：ファイル番号。ＯＰＥＮで指定した番号。１～１５の整数。

文　例　ＰＲＩＮＴ　ＬＯＣ（１）

⇨シーケンシャルアクセスファイルのときアクセスしたレコード位置が、また
ランダムアクセスファイルのとき現在のレコード番号が表示されます。

解　説

デバイス上のファイルの中の現在位置が、この関数の値になります。

シーケンシャルアクセスファイルの場合、そのファイルをオープンしてから、読み出し／書き込みを行なったレコードの位置が、関数の値となります。

ランダムアクセスファイルの場合、ＧＥＴ，ＰＵＴステートメントでその前に読み出し／書き込みが行なわれたレコード番号が、関数の値となります。

指定したデバイスが"ＣＯＭ："の場合は、受信バッファ（最大６４バイト）に入っているデータのバイト数が、関数の値として返ります。

なお、この関数はカセットでは使用できません。

参　照　3.5.2 ＬＯＦ、3.5.3 ＦＰＯＳ、

サンプルプログラム

```
10 ' LOC <例>
20 INIT:WIDTH40
30 MAXFILES1
40 N=0
50 INPUT"ファイル名";NM$
60 OPEN"R",#1,"1:"+NM$
70 FIELD#1,24 AS N$,24 AS T$,24 AS B$
80 LABEL"入力":N=N+1
90 INPUT"名前=";X$
100 IF X$="END" OR X$="end" OR X$="オワリ"
THEN "クローズ""
110 LSET N$=X$
120 INPUT"電話=";Y$
130 LSET T$=Y$
140 INPUT"住所=";Z$
150 LSET B$=Z$
160 PRINT X$;" のデータは";LOC(1);"番レ
コードに記録しました。"
170 PUT#1,N
180 GOTO"入力"
190 LABEL"クローズ""
200 CLOSE#1
210 END
```

２０：画面の初期設定を行ないます。
３０：ファイルを１つだけ使用できるように設定します。
４０：レコード番号Ｎを０に設定します。
５０：キーから入力したファイル名をＮＭ＄に入れます。
６０：ディスクのドライブ１のＮＭ＄で示されるファイルをファイル番号１番
　　のランダムアクセスファイルとしてオープンします。
８０～１８０：名前、電話番号、住所をファイルに書き込みます。
１６０：データを書き込んだレコード番号を表示します。

このプログラムは、名前をキー入力する時点で「ＥＮＤ」と入れると終了します。

3.5

3.5.2 LOF

機能 ファイルのサイズを返します。

書式

```
LOF (n)
```

n：ファイル番号。OPENで指定した番号。1～15の整数。

文例 PRINT LOF (1)

⇨ファイルのレコード数を表示します。

解説 現在オープンしているファイルのサイズが、この関数の値となります。

シーケンシャルアクセスファイルの場合、そのファイルが使用しているレコードの個数－1が関数の値となります。

ランダムアクセスファイルの場合、そのファイルで使用されているレコード番号の最大レコード番号が、関数の値となります。

指定したデバイスが"COM："の場合は、受信バッファの残りバイト数を返します。

なお、この関数はカセットでは使用できません。

参照 3.5.1 LOC

サンプルプログラム 次の例は、ディスクのファイルをコピーするプログラムです。

ドライブ0に元となるフロッピーディスクを、ドライブ1にコピーされるフロッピーディスクを入れてファイル名を指定するとコピーを開始します。

```
10 ' LOF (例)
20 INIT:WIDTH40,25
30 MAXFILES2
40 INPUT"ファイル名";NM$
50 OPEN"R",#1,"0:"+NM$
60 FIELD#1,128 AS A$,128 AS B$
70 OPEN"R",#2,"1:"+NM$
80 FIELD#2,128 AS C$,128 AS D$
90 FOR I=0 TO LOF(1)
100 GET#1,I
110 LSET C$=A$:LSET D$=B$
120 PUT#2,I
130 NEXT
140 CLOSE#1,#2
150 END
```

20：画面の初期設定をします。

30：使用できるファイル番号を1と2の2つにします。

40：コピーしたいファイル名を入力します。

90～130：ファイルを最終レコードまでコピーします。

140：ファイルをクローズします。

3.5.3 FPOS

機　　能　デバイスの先頭から数えた現在位置を返します。

書　　式

```
FPOS (n)
```

ｎ：ファイル番号。ＯＰＥＮで指定した番号。１～１５の整数。

省 略 形　ＦＰ．

文　　例　ＤＥＶＩ＄"0："，ＦＰＯＳ（１），Ａ＄，Ｂ＄

⇨３または５インチフロッピーディスクのドライブ0の現在位置するレコードから、１２８バイトずつＡ＄とＢ＄に文字データを読み込みます。

解　　説

現在アクセスしているファイルがデバイス上の何レコード目にあるかを返します。

ここでいうレコードというのは、デバイスの先頭を0レコードとして、２５６バイト単位で扱われるデータの集合をいいます。

たとえば、フロッピーではトラック0、サイド0、セクタ１を0として数えた通し番号です。

またＥＭＭ：では0番地を0レコード、ＭＥＭ：では４０００番地が0レコードとなります。

ファイルのオープン直後は、このファイルの最初の位置がデバイスの先頭から数えて何レコード目かが、この関数の値となります。ファイルをアクセスするとこの関数で与えられる位置は移動し、ファイルが最後に読み書きした位置を返します。

この関数はディスク、ＥＭＭ、ＭＥＭのみ使用可能です。

サンプルプログラム

```
10 ' FPOS (例)
20 INIT:WIDTH40,25
30 MAXFILES1
40 OPEN"O",#1,"1:TEST"
50 LABEL"入力"
60 INPUT"名前=";N$
70 IF N$="END" OR N$="end" OR N$="ｵﾜﾘ" T
HEN "ｸﾛｰｽﾞ"
80 INPUT"電話=";T$
90 INPUT"住所=";B$
100 WRITE#1,N$;T$;B$
110 PRINT
120 GOTO"入力"
130 LABEL"ｸﾛｰｽﾞ"
140 CLOSE#1
150 K$=INKEY$(1)
160 OPEN"I",#1,"1:TEST"
170 LABEL"読み出し"
180 IF LOC(1)=LOF(1) THEN "ﾃﾞｰﾀ ﾁｪｯｸ"
190 IF LOF(1)=0 THEN "ﾃﾞｰﾀ ﾁｪｯｸ"
200 X$=INPUT$(128,1):Y$=INPUT$(128,1)
210 GOTO"読み出し"
220 LABEL"ﾃﾞｰﾀ ﾁｪｯｸ"
230 DEVI$"1:",FPOS(1),X$,Y$
240 PRINT#0,X$;Y$
250 CLOSE#1
260 END
```

3.5.4　ＡＴＴＲ＄

機　能

ファイルの属性を返します。

書　式

```
ATTR$ (file)
```

file：「ファイルの指定」参照。
　　使用可能なデバイス名は、次の通りです。
　　０：～３：、ＭＥＭ０：～ＭＥＭ１：、ＥＭＭ０：～ＥＭＭ９：
　　Ｆ０：～Ｆ３：、ＨＤ０：～ＨＤ３：

省略形

ＡＴＴ．

文　例

ＰＲＩＮＴ　ＡＴＴＲ＄（"０：ＴＥＳＴ"）
⇨３または５インチフロッピーディスクのドライブ０の中のＴＥＳＴというファイルの属性を表示します。

解　説

ファイルの属性を表わす文字（Ｐ、ＲまたはＳ）が、この関数の値になります。
ファイルの属性は、ＳＥＴステートメントによってつけられ、Ｐ、Ｒ、Ｓの３つのタイプがあります。
Ｐはファイルに「ライトプロテクト（書き込み禁止）」の属性がついていることを示し、Ｒはファイルに「リードアフターライト（セーブした後、自動的にベリファイする）」の属性がついていることを、Ｓはファイルに「シークレット（ＦＩＬＥＳを実行したとき表示されない）」の属性がついていることを示しています。

参　照

2.1.32 ＳＥＴ、『ユーザーズマニュアル』の「ファイルについて」

サンプルプログラム

ＡＴＴＲ＄を使って、指定したファイルの属性を答えるプログラムを作ってみましょう。

```
10 ' ATTR$ (例)
20 INIT:WIDTH40,25
30 INPUT"ﾌｧｲﾙ名";NM$
40 PRINT"ﾌｧｲﾙ ";NM$;" の属性は「";
50 A$="解除"
60 IF ATTR$(NM$)="R" THEN A$="ﾘｰﾄﾞ ｱﾌﾀ ﾗ
ｲﾄ"
70 IF ATTR$(NM$)="P" THEN A$="ﾗｲﾄ ﾌﾟﾛﾃｸﾄ
"
80 PRINT A$+"」です。"
90 END
```

２０：画面の初期設定をします。
３０：入力されたファイル名をＮＭ＄に入れます。
４０～８０：ファイルＮＭ＄の属性を表示します。

```
RUN

ﾌｧｲﾙ名? 0:Utility
ﾌｧｲﾙ 0:Utility の属性は「ﾗｲﾄ ﾌﾟﾛﾃｸﾄ」です
。
Ok
```

3.5.5 DEVF

機　能　**デバイスの未使用領域のクラスタ数を返します。**

書　式

```
DEVF ("fd")
```

fd：デバイス名。使用可能なデバイス名は、次の通りです。
　　0：～3：、MEM0：～MEM1：、EMM0：～EMM9：、
　　F0：～F3：、HD0：～HD3：

文　例　PRINT　DEVF（"0："）
⇨3または5インチフロッピーディスクのドライブ0の残りのクラスタ数を表示します。

解　説　デバイスの未使用領域のクラスタ数（1クラスタ＝4096バイト＝4KB）が、この関数の値になります。

3.5

参　照　3.8.4 FRE／SIZE、『ユーザーズマニュアル』の「ディスクの使い方」

サンプルプログラム

```
10 ' DEVF (例)
20 INPUT"ト゛ライフ゛番号";N
30 N$=STR$(N)+":"
40 PRINT DEVF(N$)
```

20：ドライブ番号Nを入力します。
30：ドライブ番号のファイルディスクリプタを作って代入します。
40：N$の残りのクラスタ数を表示します。

```
RUN
ト゛ライフ゛番号? 0
 40
Ok
```

3.5.6　EOF

機　　能

シーケンシャルファイルの終わりを検出します。

書　　式

```
EOF (n)
```

n：ファイル番号。OPENでオープンされた番号。

文　　例

IF　EOF (1)　THEN　CLOSE　#1

⇨ファイル番号で1のファイルが終わりならばファイルをクローズします。

解　　説

ファイル番号で指定したファイルからデータを読み込んだときに、ファイルの最後のデータであれば、－1が、そうでなければ、0が、この関数の値になります。

デバイスが"COM："の場合は、EOFのコードを指定できます。詳しくは、『ユーザーズマニュアル』の「RS－233Cインターフェイスの使い方」を参照してください。

参　　照

2.4.2 MAXFILES、2.4.3 OPEN

サンプルプログラム

```
10 ' EOF (例)
20 INIT:WIDTH40
30 MAXFILES1
40 OPEN"O",#1,"1:TEST"
50 LABEL"入力"
60 INPUT"名前=";N$
70 IF N$="END" OR N$="end" OR N$="オワリ" T
HEN "クローズ"1"
80 INPUT"電話=";T$
90 INPUT"住所=";B$
100 WRITE#1,N$;T$;B$
110 PRINT
120 GOTO"入力"
130 LABEL"クローズ"1"
140 CLOSE#1
150 PRINT
160 K$=INKEY$(1)
170 OPEN"I",#1,"1:TEST"
180 LABEL"読み出し"
190 IF EOF(1) THEN "クローズ"2"
200 INPUT#1,N$,T$,B$
210 PRINT N$,T$,B$
220 GOTO"読み出し"
230 LABEL"クローズ"2"
240 CLOSE#1
250 END
```

3.6 入出力用関数

3.6.1 INP

機能

I／Oポートから読み込んだ1バイトのデータを返します。

書式

```
INP (a)
```

a：I／Oポートアドレス。0～&HFFFF。

文例

PRINT INP (&H1A01)

⇨I／Oポートのアドレス&H1A01から読み込んだ1バイトのデータが表示されます。

解説

I／Oポートのアドレスaから読み込んだ1バイトのデータが、この関数の値になります。

関数の値は、0～255の整数です。

参照

2.3.13 OUT、3.6.3 PEEK@

サンプルプログラム

```
10 ' INP (例)
20 SOUND 7,&H38
30 OUT &H1C00,7
40 PRINT HEX$(INP(&H1B00))
```

SOUND7、&H38を実行した後、I／Oポートの1C00番地にレジスタ番号7を出力すると、1B00番地にはデータの&H38が書き込まれます。

```
RUN
38
Ok
```

3.6.2 PEEK

機　能

メインメモリ内の指定したアドレスから読み込んだ１バイトのデータを返します。

書　式

```
PEEK (a)
```

a：メインメモリのアドレス。0～&HFFFF。

省略形

PE.

文　例

PRINT　PEEK（&HC000）

⇨メインメモリ内のアドレス&HC000から取り出した１バイトのデータが表示されます。

解　説

メインメモリ内の指定したアドレスから取り出した１バイトのデータが、この関数の値になります。

関数の値は、0～255の整数です。

参　照

2.3.13 OUT、3.6.3 PEEK@

サンプルプログラム

PALETステートメントを実行したときのF8D2、F8D3、F8D4番地の内容をPEEK関数を使ってみましょう。

```
10 ' PEEK (例)
20 INIT:WIDTH40,25
30 PALET
40 A$="PALET":GOSUB120
50 PALET 0,1
60 A$="PALET 0,1":GOSUB120
70 PALET 0,3
80 A$="PALET 0,3":GOSUB120
90 PALET
100 A$="PALET":GOSUB120
110 END
120 PRINT A$
130 PRINT"&HF8D2: ";BIN$(PEEK(&HF8D2))
140 PRINT"&HF8D3: ";BIN$(PEEK(&HF8D3))
150 PRINT"&HF8D4: ";BIN$(PEEK(&HF8D4))
160 PRINT
170 PAUSE20
180 RETURN
```

3.6.3 PEEK@

機　能　**ＶＲＡＭ内の指定したアドレスから読み込んだ１バイトのデータを返します。**

書　式

```
PEEK@(a)
```

ａ：ＶＲＡＭ（Ｉ／Ｏポートの一部）内のアドレス。
　　＆Ｈ２０００～＆Ｈ２７ＦＦ、＆Ｈ３０００～＆ＨＦＦＦＦ。

省略形　ＰＥ．＠

文　例　ＰＲＩＮＴ　ＰＥＥＫ＠(＆Ｈ３０００)

⇨テキスト画面の現在のホーム位置の属性を表わす１バイトのデータが表示されます。

解　説

６４Ｋバイト存在するＩ／Ｏポートのうち、＆Ｈ３０００～＆Ｈ３７ＦＦの２ＫバイトはテキストＶＲＡＭ（Ｖｉｄｅｏ　Ｒａｎｄｏｍ　Ａｃｃｅｓｓ　Ｍｅｍｏｒｙ）、＆Ｈ２０００～＆Ｈ２７ＦＦの２Ｋバイトはその属性エリア、＆Ｈ３８００～＆Ｈ３ＦＦＦは全角文字の属性エリア、＆Ｈ４０００～＆ＨＦＦＦＦはグラフィックエリアとして使用されています。

ＰＥＥＫ＠関数は、このＶＲＡＭのアドレスａに入っている１バイトのデータを値としてもちます。

関数の値は、０～２５５の整数です。

参　照　2.7.10 ＰＯＫＥ＠、3.6.2ＰＥＥＫ、
『ユーザーズマニュアル』の「テキスト画面（ＶＲＡＭ）へのアクセス」

サンプルプログラム

次の例は、テキスト画面の状態を表示示するプログラムです。

```
10 ' PEEK@ <例>
20 INIT:WIDTH40,25
30 FOR I=0 TO 7
40 COLOR I:PRINT CHR$(&H40+I);
50 NEXT
60 PRINT
70 FOR I=0 TO 7
80 PRINT PEEK@(&H3000+I),
90 B$=BIN$(PEEK@(&H2000+I))
100 PRINT RIGHT$("00000000"+B$,8)
110 NEXT
120 END
```

２０：画面を初期設定します。
３０～５０：テキスト画面に＆Ｈ４０～＆Ｈ４７のキャラクタコードを持つ文字「＠ＡＢＣＤＥＦＧ」を表示します。
７０～１１０：テキスト画面とその属性の状態を表示します。

```
RUN

@ABCDEFG
 64         00000000
 65         00000001
 66         00000010
 67         00000011
 68         00000100
 69         00000101
 70         00000110
 71         00000111
Ok
```

※実際の画面では@は見えません。

3.6

3.7 ジョイステック関数

3.7.1 STICK

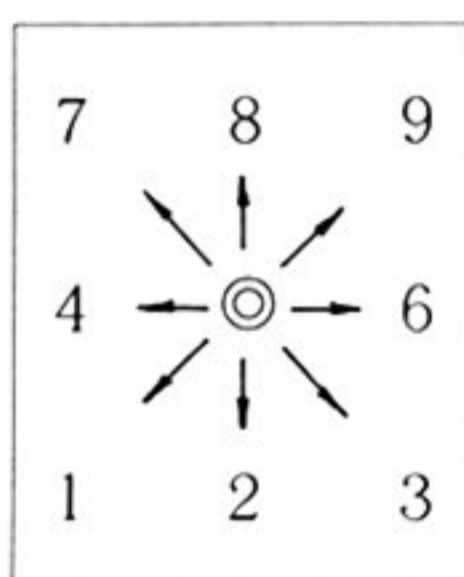
機能　**ジョイスティックまたはテンキーの状態を返します。**

書式

```
STICK (n)
```

n：ジョイスティックの番号。0～2の整数。
　0：テンキー。
　1：ジョイスティック1。
　2：ジョイスティック2。

省略形　STI.

解説　ジョイスティックまたはキーボードのテンキーから入力された値が、この関数の値になります。
ジョイスティックを接続している場合は、スティックを倒す方向によって、下の図のような値となります。

スティックを倒さなければ、0となります。
キーボード上のテンキーの1～9の値を押すと、その値が返ります。それ以外のキーが押されると、0となります。なお、メインキーボード上の1～9のキー押しても1～9の値となります。

参照　3.7.2 STRIG

3.7

サンプルプログラム

テンキーによって画面上のキャラクタを移動するプログラムを作ってみましょう。

```
10 ' STICK (例)
20 INIT:WIDTH 40,25,0
30 X=20:Y=12:I=20:J=12
40 S=STICK(0)
50 IF S=3 OR S=6 OR S=9 THEN X=X+1
60 IF S=1 OR S=4 OR S=7 THEN X=X-1
70 IF S=7 OR S=8 OR S=9 THEN Y=Y-1
80 IF S=1 OR S=2 OR S=3 THEN Y=Y+1
90 IF X>38 THEN X=38
100 IF X<0  THEN X=0
110 IF Y>22 THEN Y=22
120 IF Y<0  THEN Y=0
130 LOCATE I,J:PRINT" "
140 LOCATE X,Y:PRINT"O"
150 I=X:J=Y
160 GOTO40
```

２０：画面の初期設定を行ないます。

３０：キャラクタの位置の初期値を与えます。

４０：テンキーの状態を変数Ｓに代入します。

５０～１２０：Ｓの値によってＸとＹの座標を移動します。９０～１２０はキャラクタが画面の外へ出ないようにするためのクリッピングです。

１３０：前の位置のキャラクタを消します。

１４０：新しい位置にキャラクタを表示します。

１６０：４０番に戻って次のテンキーの状態をみます。

3.7

3.7.2 STRIG

機　　能

ジョイスティックのトリガーボタンまたはキーボードのスペースキーの状態を返します。

書　　式

```
STRIG (n)
```

n：入力の対象を表わす。0～2の整数。

0：スペースキー。

1：ジョイスティック1。

2：ジョイスティック2。

省略形

STR.

解　　説

ジョイスティックのトリガーボタン、またはキーボードのスペースキーが、押されていれば、－1が、押されていなければ、0が、この関数の値になります。

参　　照

3.7.1 STICK

サンプルプログラム

STICKのサンプルプログラムにSTRIGステートメントを付け加えてみましょう。

```
10 ' STRIG (例)
20 INIT:WIDTH 40,25,0
30 X=20:Y=12:I=20:J=12
40 S=STICK(0)
50 IF S=3 OR S=6 OR S=9 THEN X=X+1
60 IF S=1 OR S=4 OR S=7 THEN X=X-1
70 IF S=7 OR S=8 OR S=9 THEN Y=Y-1
80 IF S=1 OR S=2 OR S=3 THEN Y=Y+1
85 IF STRIG(0)=-1 THEN COLOR ,2:BEEP:COL
OR ,0
90 IF X>38 THEN X=38
100 IF X<0  THEN X=0
110 IF Y>22 THEN Y=22
120 IF Y<0  THEN Y=0
130 LOCATE I,J:PRINT" "
140 LOCATE X,Y:PRINT"O"
150 I=X:J=Y
160 GOTO40
```

85：キーボードのスペースキーが押されたら、背景色を赤にしてビープ音を出します。

3.7

3.8 特殊関数

3.8.1 ＰＯＳ

機能

カーソル横方向の現在位置を返します。

書式

```
POS (n)
```

ｎ：ファイル番号。ＭＡＸＦＩＬＥＳで指定した値まで。０～１５の整数。０のときは画面を指定。

文例

Ｘ＝ＰＯＳ（０）
⇨テキスト画面上の現在のカーソルの水平位置の値がＸに代入される。

解説

ｎ＝０を指定すると画面上の現在のカーソルの横方向の位置が、この関数の値になります。

このときは、画面の左端を０とする整数の値をもち、その範囲は、ＷＩＤＴＨ８０のとき０～７９、ＷＩＤＴＨ　４０のとき、０～３９です。

カーソルのたて方向の位置は、ＣＳＲＬＩＮ変数の値によって知ることができます。

ｎに１、２、……の整数を指定すると、シーケンシャルアクセスファイルにＰＲＩＮＴ＃やＷＲＩＴＥ＃で文字を書き込んでいるとき、その文字が何バイト目にあたるかを示す値がこの関数の値になります。改行コードが送られると、値は０に戻ります。

参照

3.10.1 ＣＳＲＬＩＮ、3.8.2 ＬＰＯＳ

サンプルプログラム

次は、画面にたて横２倍の拡大文字を表示するためのプログラムで、カーソルはたて横２倍文字に合わせて２文字ずつ移動するようになっています。

```
10 ' POS (例)
20 INIT:WIDTH 40,25,0:CSIZE3:CLS
30 X=0:Y=0
40 LOCATE X,Y
50 K$=INKEY$(1)
60 IF K$>=" " THEN PRINT#0,K$;:GOTO 80
70 PRINT K$;K$;
80 X=POS(0):Y=CSRLIN
90 GOTO40
```

２０：画面の初期設定と拡大文字を表示する設定を行ないます。

３０：カーソル位置を示す変数ＸとＹの初期化を行ないます。

４０：カーソル位置を設定します。

５０：カーソルをブリンキングしてキーの入力を待ちます。

６０：入力した文字が図形文字（キャラクターコードが３２より大きい）のとき、画面にその拡大文字を表示して８０番にジャンプします。

７０：入力した文字が制御文字（カーソルキーなど）のときはカーソルを２文字分スキップします。

８０：現在のカーソル位置をＸとＹに代入します。

９０：４０番に戻ります。

3.8

3.8.2 LPOS

機　能　プリンタのヘッドの位置を返します。

書　式

```
LPOS (n)
```

n：0、1、2の整数。

文　例　PRINT　LPOS（0）

⇨現在のプリンタのヘッドの位置が表示されます。

解　説

プリンタのヘッドの位置が、この関数の値になります。

正確にいえば、プリンタバッファ内のヘッドの位置がどこにあるかを示します。したがって、実際のプリンタのヘッドの現在位置を示すとは限りません。

nの値によってLPOSは次のような値を返します。

nの値	LPOSの値
0	印字したすべての文字をカウントしたヘッドの位置
1	印字した文字のうち半角文字のみカウントしたヘッドの位置
2	印字した文字のうち全角文字のみカウントしたヘッドの位置

関数の値は、0～240の整数です。

この関数を使って、簡単なプリント書式の制御をすることができます。

参　照　3.5.3 FPOS

3.8

3.8.3 VARPTR

機　　能　　変数が格納されているアドレスを返します。

書　　式

〔1〕VARPTR（x）
〔2〕VARPTR（x$）
〔3〕VARPTR（#n）

x：調べたい数値変数。
x$：調べたい文字変数。
n：調べたいファイル番号。1～15の整数。MAXFILESで指定された値まで。

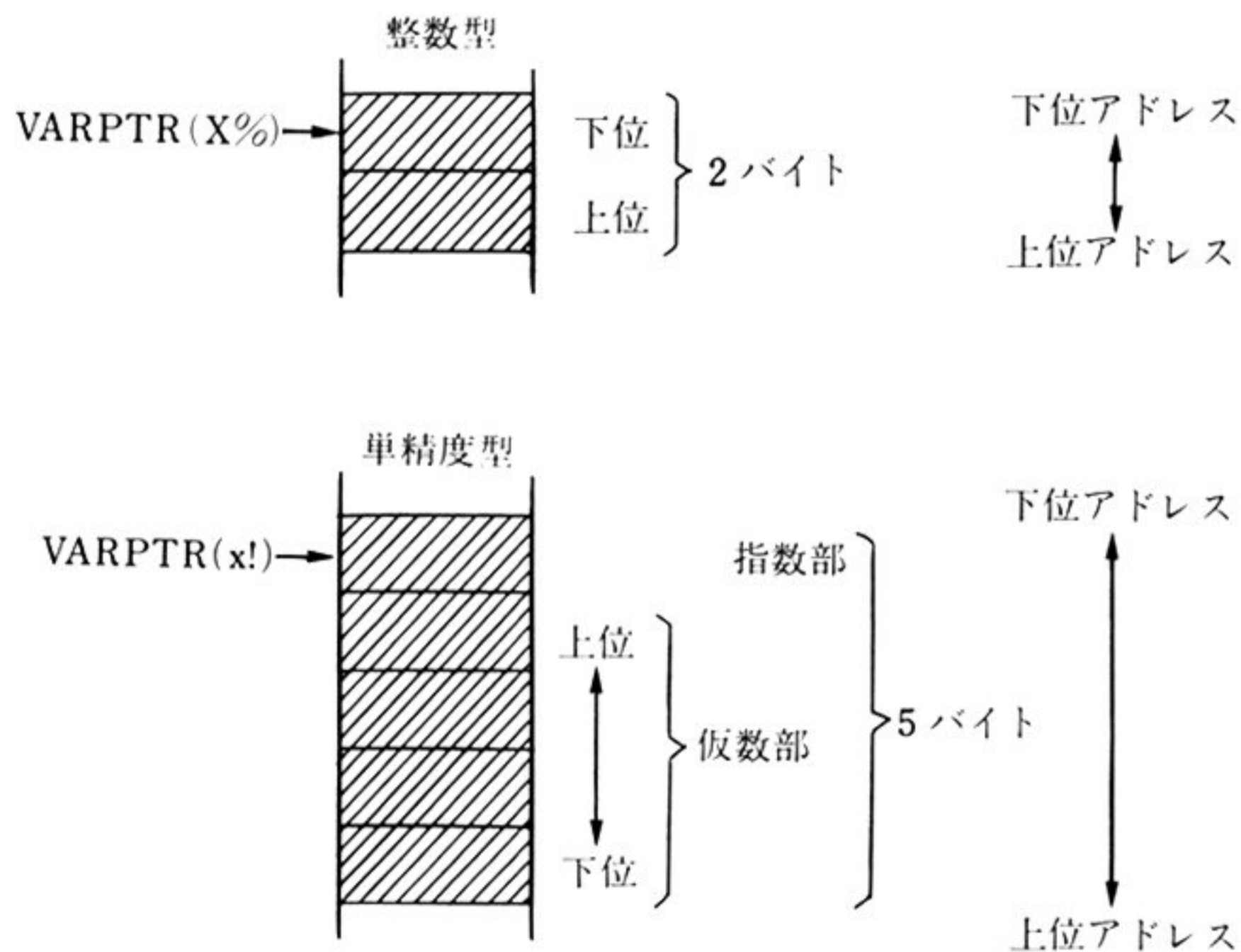
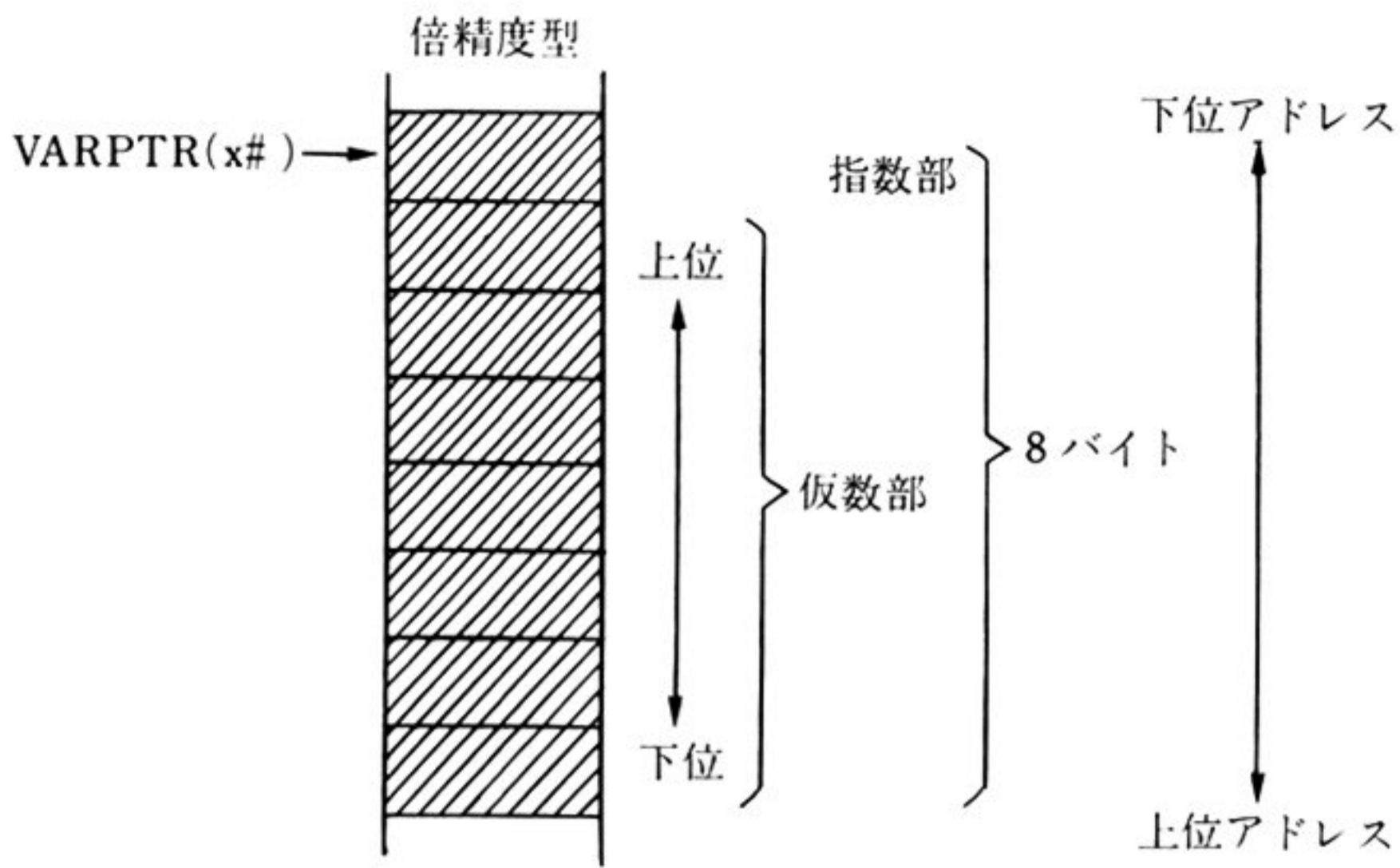
省 略 形　　VAR.

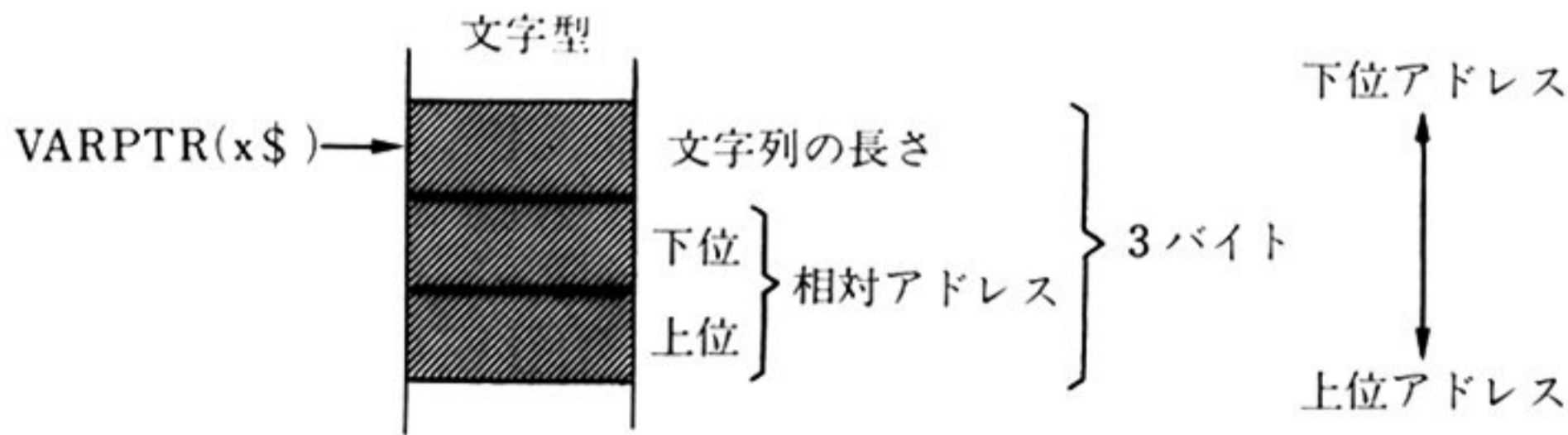
文　　例　　A＝VARPTR（X）
⇨変数Xの内容が格納されているメモリの先頭アドレスがAに代入されます。

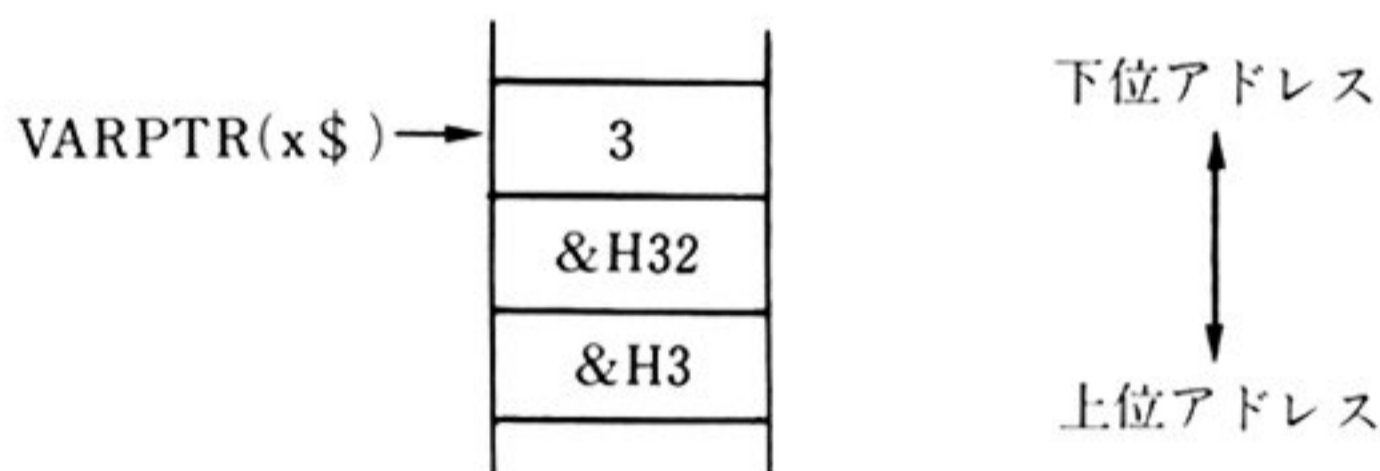
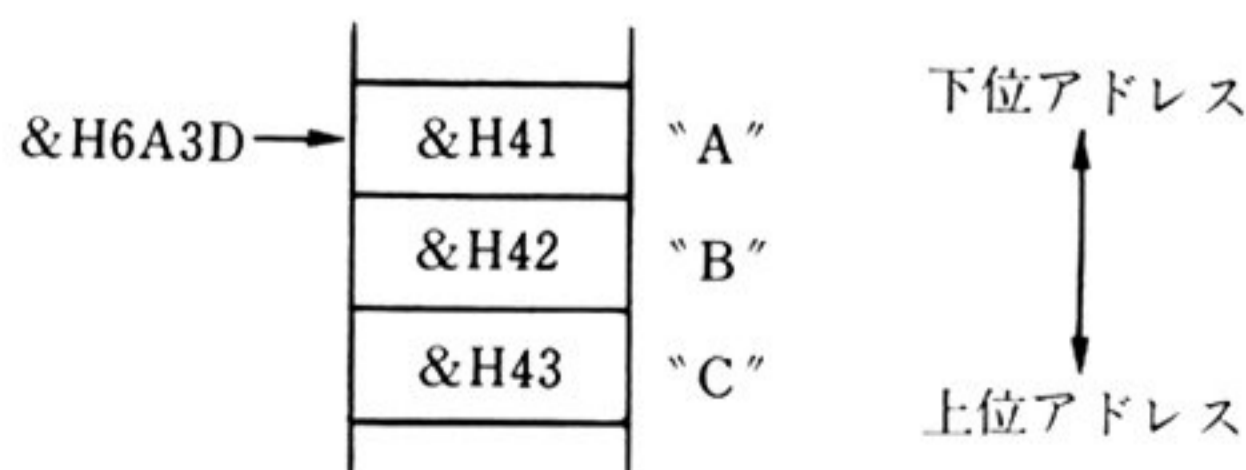
解　　説

VARPTRは、変数の格納されているアドレスを返す関数で、数値変数、文字変数あるいはファイル番号の場合で返される値の意味が異なります。

〔1〕数値変数Xを指定した場合
数値変数Xの値が格納されている変数エリアの先頭アドレスが、返されます。

3.8

〔2〕文字変数X＄を指定した場合

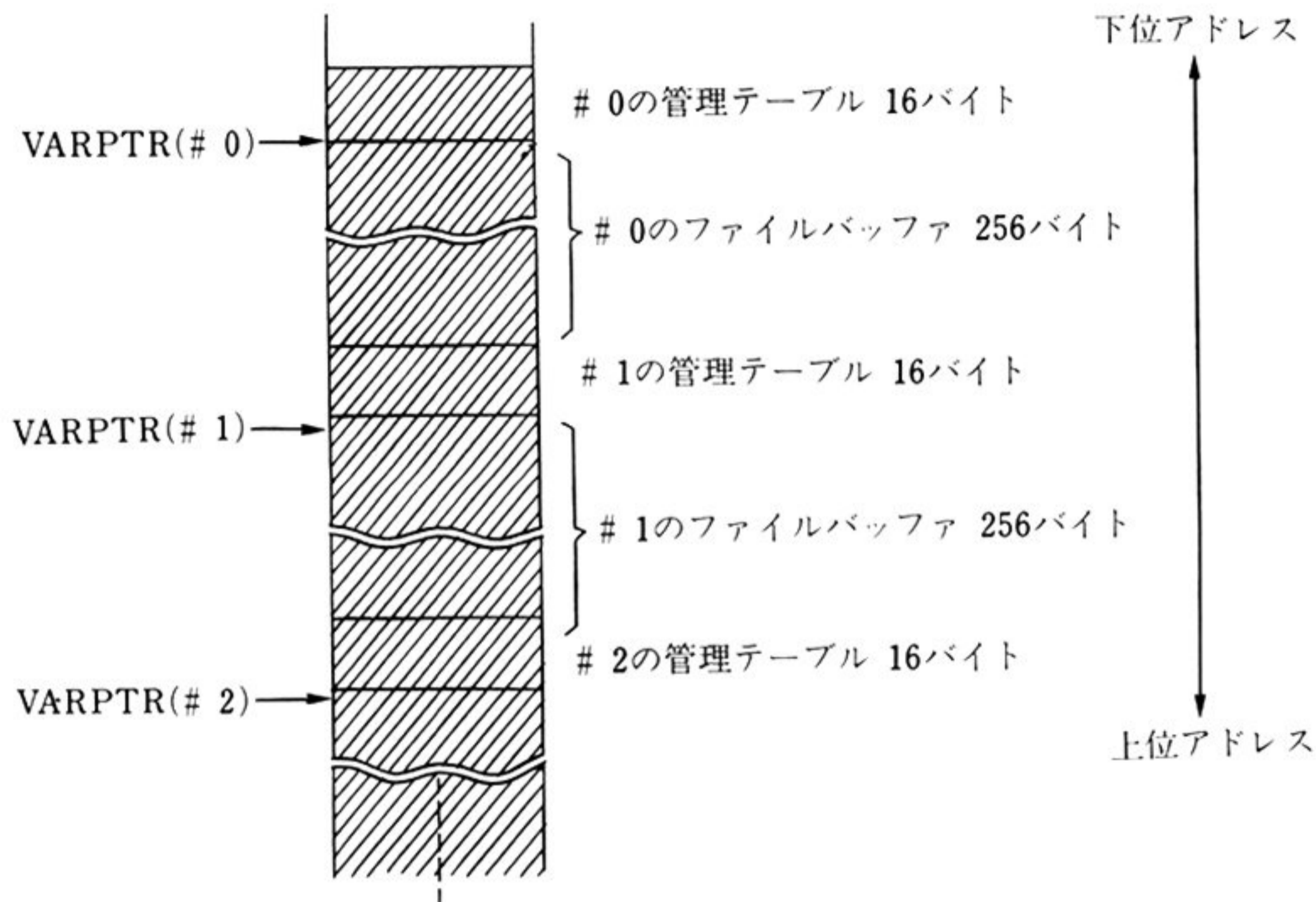
文字変数X＄の値（文字列）が格納されているストリングデータエリアの先頭アドレスを返さず、ストリングディスクリプタと呼ばれる部分の先頭アドレスが返されます。ストリングディスクリプタは、3バイトから成り、最初の1バイトが文字列の長さ、後ろの2バイトが文字列を格納してる相対アドレスを示しています。

相対アドレスとは、文字変数の値（文字列）の格納されているストリングデータエリアの先頭アドレスを0としたとき、何バイト離れたところに文字列の先頭があるかを示す値です。ストリングデータエリアの先頭アドレスはＳＴＲＰＴＲ変数に入っているので、X＄の値の格納されているアドレスは、ＳＴＲＰＴＲ変数の値に相対アドレスを加えて求めることができます。

たとえば、X＄＝"ＡＢＣ"で、ＳＴＲＰＴＲ変数の値が＆Ｈ６７０Ｂ、

のとき、X＄の値の格納されているアドレスは＆Ｈ６７０Ｂ＋＆Ｈ３３２＝＆Ｈ６Ａ３Ｄとなり、文字列"ＡＢＣ"は次のように＆Ｈ６Ａ３Ｄ番地から格納されていることになります。

ＵＳＲ関数を使って機械語サブルーチンに移るとき、引数にＶＡＲＰＴＲ（Ｘ）を指定すると、変数Ｘの値のアドレスを調べるので便利です。

〔3〕ファイル番号#nを指定した場合

ファイル番号nのファイルバッファの先頭アドレスが返されます。ファイルバッファとは、ＯＰＥＮステートメントでファイルをオープン後、ファイルにデータを読み書きするときに一時的に使われるメインメモリ上のエリアで、ＭＡＸＦＩＬＥＳで指定された数の分だけ、確保されます。

ファイル用ストリングバッファの構造

3.8

参　照　3.10.2 ＳＴＲＰＴＲ

サンプルプログラム

(例１)次に、数値変数の値の格納されている先頭アドレスを求めるプログラムを示します。

```
10 ' VARPTR (例1)
20 INIT:WIDTH 40,25
30 INPUT"X=";X
40 INPUT"Y=";Y
50 A=VARPTR(X)
60 B=VARPTR(Y)
70 PRINT"X:  ";HEX$(A)
80 PRINT"Y:  ";HEX$(B)
```

２０：画面の初期化を行ないます。

３０：４０：変数ＸとＹに値を入力します。

５０：Ｘの値が格納されているアドレスを求めてＡに代入します。このときＸの値は単精度型なのでアドレスＡから５バイト(Ａ＋４まで)のメモリに格納されています。

６０：Ｙの値が格納されているアドレスを求めてＢに代入します。

７０、８０：アドレスＡとアドレスＢを１６進数表現で表示します。

```
RUN

X=? 1
Y=? 2
X: 6639
Y: 6641
Ok
```

（例２）文字変数の値（文字列）の格納されているアドレスを求めるプログラムを示します。

```
10 ' VARPTR (例2)
20 INIT:WIDTH 40,25
30 INPUT"X$=";X$
40 A=VARPTR(X$)
50 L=PEEK(A):A1=PEEK(A+1):A2=PEEK(A+2)
60 L$=RIGHT$("00"+HEX$(A1),2)
70 H$=RIGHT$("00"+HEX$(A2),2)
80 PRINT"長さ      =";L
90 PRINT"相対アドレス=";H$;L$
100 ADR=STRPTR+VAL("&H"+H$+L$)
110 PRINT"STRPTR    =";HEX$(STRPTR)
120 PRINT"格納アドレス=";HEX$(ADR)
```

２０：画面の初期設定をします。

３０：変数Ｘ＄に文字列を入力します。

４０：Ｘ＄の情報を記憶しているアドレスをＡに代入します。

５０：文字列の長さのデータをＬ、相対アドレスの下位バイトをＡ１、上位バイトをＡ２に代入します。

６０、７０：相対アドレスを１６進数に変換して、下位バイトをＬ＄に上位バイトをＨ＄に代入します。

3.8

3.8.4 FRE／SIZE

機　能

メモリのユーザエリアの未使用部分のサイズを返します。

書　式

```
〔1〕FRE (n)
〔2〕SIZE
```

n：数式。0、1、2。単独の定数、数値変数でもかまいません。

文　例

ＰＲＩＮＴ　ＦＲＥ（0）
⇨全メモリのうち、未使用エリアのバイト数が表示されます。

解　説

メモリのユーザーエリアのうち、ＢＡＳＩＣのプログラムで使っていない部分のバイト数が、この関数の値になります。
指定するｎの値によって、次の部分のフリーエリアのバイト数が返ります。

ｎの値	内　　容
0	メモリ全体のフリーエリア
1	メインメモリのフリーエリア
2	ＶＤＩＭエリアのフリーエリア

ＳＩＺＥは、ＦＲＥ（0）の値を返します。
なお、ＮＥＷＯＮステートメントの実行によってメインメモリのフリーエリアは変化します。

3.8

参　照

3.5.5 ＤＥＶＦ、2.1.2 ＮＥＷＯＮ

サンプルプログラム

（例２）は（例１）より「２0　ＰＲＩＮＴ」の１行が多い分だけメモリが少なくなっています。

（例１）

```
10 ' FRE/SIZE (例)
20 PRINT FRE(0);"ﾊﾞｲﾄ"

RUN
 30416 ﾊﾞｲﾄ
Ok
```

（例２）

```
10 ' FRE/SIZE (例2)
20 PRINT
30 PRINT FRE(0);"ﾊﾞｲﾄ"

RUN

 30410 ﾊﾞｲﾄ
Ok
```

3.8.5 KANJI$

機能　漢字ROMの漢字パターンデータを返します。

書式

```
KANJI$ (c)
```

c：区点コード。

省略形　KA.

文例　PATTERN−16, KANJI$ (1606)

⇨区点コード1606の漢字「愛」を表示します。
(表示位置はPOSITIONで指定する)

解説　内蔵の漢字ROMから読み込んだ漢字パターン（たて16×横16ドットのサイズ）のコードが、この関数の値になります。

KANJI$はPATTERNステートメントといっしょに使用します。

参照　2.8.11 PATTERN、付録の「非漢字およびJIS第一水準漢字一覧表」

3.8

3.8.6 CMT

機　能　カセットの状態を返します。

書　式

〔1〕CMT
〔2〕CMT (n)

n：0、1、2の整数

文　例　IF CMT (1) =0 THEN PRINT"テープをセットしてください"

⇨テープがセットされていなければ「テープをセットしてください」と表示します。

解　説　CMTは、カセットの現状態を知らせる関数です。

〔1〕引数が省略されている場合、カセットの状態によって次のような値を返します。

カセットの状態	CMTの値
EJECT	0
STOP	1
READ	2
FF	3
REW	4
WRITE	10

〔2〕CMT (n) の書式の場合、カセットの状態によって次のような値を返します。

nの値	カセットの値	CMT (n) の値
0	テープが回転中	−1
	テープが停止中	0
1	テープがセットされている	−1
	テープがセットされていない	0
2	テープの録音防止用のツメを折っていない	−1
	テープの録音防止用のツメを折ってある	0

参　照　2.15.6 CMT

3.8.7　ＦＮ

機　　能　**関数名と引数を与えて、関数ルーチンを呼び出します。**

書　　式

ＦＮ関数名〔(x_1〔，x_2〕……)〕

関数名：ユーザーの定義した関数の名前。
x_1，x_2，……：ＤＥＦ　ＦＮで定義した関数の引数。

文　　例　Ｘ＝ＦＮＡ（1）
⇨引数１でユーザー定義関数ＦＮＡを計算した結果をＸに代入します。

解　　説　ＤＥＦ　ＦＮステートメントによって登録された関数の名前は、"ＦＮ"とこれに続く名前を含めたものです。名前は変数名と同様に扱われます。

参　　照　2.2.24 ＤＥＦ　ＦＮ

3.8.8　ＵＳＲ

機　　能　**引数を与えて、機械語サブルーチンを呼び出します。**

書　　式

ＵＳＲ　ｎ　(ｘ)

ｎ：ユーザー関数識別番号。ＤＥＦ　ＵＳＲで定義した番号。０～９の整数。
ｘ：ユーザー関数に引き渡すデータ。引数。

文　　例　Ｘ＝ＵＳＲ１（2）
⇨引数２をもってユーザー関数１番の機械語サブルーチンを呼び出します。
機械語サブルーチンからリターンした後、その結果をもつ関数となります。

解　　説　引数を機械語サブルーチン（ユーザー関数）に渡して、サブルーチンを呼び出します。
引数を引き渡すことができる点で、ＣＡＬＬステートメントとは区別されます。

参　　照　『ユーザーズマニュアル』の「機械語サブルーチンとモニタ」

3.8.9 CALC

機　能　ユーザーによって与えられた式または関数の演算をします。

書　式

```
CALC (X$)
```

X＄：引用符で囲まれた数式、関数。あるいは文字式。

文　例　10 PRINT CALC ("SQR (2) ")

⇨SQR（2）を計算して、1．4142136を表示します。

解　説　CALCは、ユーザーによって与えられた式または関数を演算するための特殊関数です。

引用符（"）で囲まれた部分が数式や関数のときにはその計算結果を返し、文字式のときはそれの表わす文字列を返します。

ただし、CALCはダイレクトモードで実行することができません。

例）
```
10:DIM A (10)
20:FOR I=0 TO 10
30:INPUT A$:A (I) =CALC (A$)
40:NEXT
50:FOR I=0 TO 10
60:PRINT A (I)
70:NEXT
80:END
```

このプログラムを実行すると、連続してキーから入力できます。こういったときに、数式の計算結果を入力したい場合は、そのままの数式を入力すれば、内部で計算し、結果を配列に入れてくれます。わざわざ計算をして入力しなくてすみます。

サンプルプログラム

```
10 ' CALC (例)
20 LINPUT A$
30 IF A$="END" THEN END
40 C=CALC(A$)
50 PRINT C
60 GOTO20


RUN
20*30
 600
SQR(5)
 2.236068
SIN(RAD(30))^2+COS(RAD(30))^2
 1
END
Ok
```

3.8

3.9 タイマー変数

3.9.1 TIME$

機能

時刻の設定、またはその値を与えます。

書式

```
〔1〕TIME$
〔2〕TIME$="hh:mm:ss"
```

"hh:mm:ss"：時・分・秒を表す文字列。

hh：時。00〜23（時）。

mm：分。00〜59（分）。

ss：秒。00〜59（秒）。

解説

コンピュータに内蔵されている時計の現在時刻が、この変数の値になります。

（例）PRINT　TIME$↵

12：15：30　　……現在時刻が12時15分30秒であることを示しています。

〔2〕の代入文によって、現在時刻を再設定することができます。

（例）TIME$="19：00：00"……現在時刻を19時ちょうどに設定しています。

参照

3.9.2 DAY$、3.9.3 DATE$、3.9.4 TIME

サンプルプログラム

例として、秒単位に音の出る時計を作ってみましょう。

```
10 ' TIME$ (例)
20 INIT:WIDTH40,25
30 T$=TIME$
40 IF RIGHT$(T$,2)="00" THEN MUSIC"O6A0"
 ELSE MUSIC"O5A0"
50 LOCATE0,0
60 PRINT"時刻: ";TIME$
70 IF T$=TIME$ THEN 50
80 GOTO30
```

20：画面の初期化を行ないます。

30：TIME$の時刻をT$にコピーします。

40：音を出します。

50：60：現在時刻を表示します。

70：TIME$がT$に等しいうちは時刻の表示だけを続けます。

80：TIME$が変わって(すなわち秒が変わって)T$と等しくなくなったら30番に戻ってTIME$をT$にコピーしなおして音を出します。

現在時刻を正確に合わせたいときは次の例のような時刻の設定を時報とともに行なってください。

```
TIME$="12:00:00"
Ok
```

3.9.2 DAY$

機能

曜日の設定、またはその値を与えます。

書式

〔1〕**DAY$**
〔2〕**DAY$="XXX"**

"XXX"：曜日を表す文字列。次の略記号で表されます。

SUN：日曜日（Sunday）。
MON：月曜日（Monday）。
TUE：火曜日（Tuesday）。
WED：水曜日（Wednesday）。
THU：木曜日（Thursday）。
FRI：金曜日（Friday）。
SAT：土曜日（Saturday）。

解説

コンピュータに内蔵されているカレンダーの今日の曜日が、この変数の値になります。

(例) PRINT　DAY$⏎
　　SUN　　　　　……今日が日曜日であることを示しています。

〔2〕の代入文によって、今日の曜日を再設定することができます。

(例) DAY$="MON"……今日の曜日を月曜日に設定しています。

参照

3.9.1 TIME$、3.9.3 DATE$、3.9.4 TIME

サンプルプログラム

TIME$のサンプルプログラムに曜日の表示をつけ加えましょう。

```
10 ' DAY$ (例)
20 INIT:WIDTH40,25
30 T$=TIME$
40 IF RIGHT$(T$,2)="00" THEN MUSIC"O6A0"
 ELSE MUSIC"O5A0"
50 LOCATE0,0
60 PRINT"曜日: ";DAY$
70 PRINT"時刻: ";TIME$
80 IF T$=TIME$ THEN 50
90 GOTO30
```

60：曜日を表示します。

曜日が違う場合は次の例のように設定しなおしてください。

```
DAY$="SUN"
Ok
```

3.9

3.9.3 DATE$

機　能　年月日の設定、またはその値を与えます。

書　式

〔1〕DATE$
〔2〕DATE$="yy/mm/dd"

"yy／mm／dd"：年・月・日を表す文字列。
yy：年。00～99（年）。
mm：月。01～12（月）。
dd：日。01～31（日）。

解　説

コンピュータに内蔵されているカレンダーの今日の年月日が、この変数の値になります。

(例) PRINT　DATE$⏎
85／10／06　　……今日の年月日が1985年10月6日であることを示しています。

〔2〕の代入文によって、今日の年月日を再設定することができます。

(例) DATE$="85／05／24"
…… 今日の年月日を1985年5月24日に設定しています。

※またディスク　BASICの場合、Start upプログラムDATE$の年を設定するプログラムがはいっていますので、現在の年にこのプログラムの一部を書きかえてください。

参　照　3.9.1 TIME$、3.9.2 DAY$、3.9.4 TIME

サンプルプログラム

TIME$のサンプルプログラムに年月日と曜日の表示をつけ加えて、カレンダー時計を作ってみましょう。

```
10 ' DATE$ (例)
20 INIT:WIDTH40,25
30 T$=TIME$
40 IF RIGHT$(T$,2)="00" THEN MUSIC"O6A0"
 ELSE MUSIC"O5A0"
50 LOCATE0,0
60 PRINT"年月日: ";DATE$
70 PRINT"  曜日: ";DAY$
80 PRINT"  時刻: ";TIME$
90 IF T$=TIME$ THEN 50
100 GOTO30
```

60：年月日を表示します。

年月日の年が？？となっているときは次の例のように設定しなおしてください。

```
DATE$="85/01/01"
Ok
```

3.9

3.9.4 TIME

機能　秒数の設定、またはその値を与えます。

書式

〔1〕TIME
〔2〕TIME＝s

s：秒数。０～８６３９９の整数。

解説　ＴＩＭＥの初期値に経過した時間（秒数）を加えた値が、この変数の値となります。

ＴＩＭＥの初期値には、０～８６３９９の整数を当てることができます。８６３９９は２３時間５９分５９秒を秒数に換算した値です。

ＴＩＭＥの値は、８６３９９を越えると再び０に戻ります。

ＴＩＭＥの変数は、ストップウォッチのように経過時間を測るのに使うことができます。

参照　3.9.1 ＴＩＭＥ＄

サンプルプログラム　ＴＩＭＥを使って、ＬＩＮＥのボックスフィルのスピードを計ってみましょう。

```
10 ' TIME (例)
20 INIT:WIDTH40,25
30 TIME=0
40 FOR I=1 TO 7
50 LINE(0,0)-(319,199),PSET,I,BF
60 NEXT
70 CLS4
80 PRINT TIME;"秒"
```

２０：画面の初期設定を行ないます。

３０：ＴＩＭＥを０に設定します。

４０～６０：パレットコード１～７のボックスフィルを行ないます。

3.9

3.10 システム変数

3.10.1 CSRLIN

機能　画面上のカーソルが何行目にあるかを返します。

書式

```
CSRLIN
```

省略形　CSR.

文例　PRINT CSRLIN
⇨カーソルの垂直位置を表示します。

解説　CSRLINは、カーソルが、現在、画面最上位を0としたとき、何行目にあるかを示すシステム変数です。変数の値の範囲は、テキスト10行のとき0～9、テキスト12行のとき0～11、テキスト20行のとき0～19、テキスト25行のとき0～24です。

参照　3.8.1 POS

サンプルプログラム　次の例は、画画にたて横2倍文字を順次表示するためのプログラムで、カーソルが2文字分ずつスキップするように作られています。

```
10 ' CSRLIN <例>
20 INIT:WIDTH 40,25,0:CSIZE3:CLS
30 X=0:Y=0
40 LOCATE X,Y
50 K$=INKEY$(1)
60 IF K$>=" " THEN PRINT#0,K$;:GOTO 80
70 PRINT K$;K$;
80 X=POS(0):Y=CSRLIN
90 GOTO40
```

20：初期画面の設定とたて横の2倍の拡大文字を表示する設定をします。
30：カーソル位置を指定する変数X、Yの初期化を行ないます。
40：カーソル位置を設定します。
50：カーソルをブリンキングしてキーの入力待ちとなります。
60：入力された文字が図形文字(キャラクタコードが32より大きい)のとき画面にその文字を表示して80番にジャンプします
70：入力された文字が制御文字(カーソルキーなど)のときはカーソルを2文字分スキップします。
80：現在カーソルの位置をX、Yに代入します。
90：40に戻ります。

3.10

3.10.2 STRPTR

機　能

文字変数の値に記憶されているエリアの先頭アドレスを返します。

書　式

```
STRPTR
```

省略形

ＳＴＲＰ．

文　例

ＰＲＩＮＴ　ＨＥＸ＄（ＳＴＲＰＴＲ）
⇨文字変数の格納されているメモリ内のアドレスを１６進数で表示します。

解　説

数値変数を格納しているアドレスは、ＶＡＲＰＴＲ関数の値を直接読んで求めることができますが、文字変数のときは同様に求めることができません。

ＳＴＲＰＴＲには、文字変数の格納されているメインメモリ内のエリアの先頭アドレスが入っており、また、ＶＡＲＰＴＲには、先頭アドレスを０としたとき、そこから何バイト離れたところに指定された文字変数があるかを示す相対アドレスと文字変数の文字列の長さを知るための情報が格納されているアドレスが入っています。

したがって、文字変数を格納しているアドレスは、ＳＴＲＰＴＲの値にＶＡＲＰＴＲ関数の値をたして求めることができます。

求め方の詳細はＶＡＲＰＴＲを参照してください。

参　照

3.8.3 ＶＡＲＰＴＲ

サンプルプログラム

文字変数を格納しているアドレスを求めるプログラムを次に示します。

```
10 ' STRPTR (例)
20 INIT:WIDTH 40,25
30 INPUT"X$=";X$
40 A=VARPTR(X$)
50 L=PEEK(A):A1=PEEK(A+1):A2=PEEK(A+2)
60 L$=RIGHT$("00"+HEX$(A1),2)
70 H$=RIGHT$("00"+HEX$(A2),2)
80 PRINT"長さ       =";L
90 PRINT"相対ｱﾄﾞﾚｽ=";H$;L$
100 ADR=STRPTR+VAL("&H"+H$+L$)
110 PRINT"STRPTR    =";HEX$(STRPTR)
120 PRINT"格納ｱﾄﾞﾚｽ=";HEX$(ADR)
```

２０：画面の初期化を設定します。
３０：入力した文字列を変数Ｘ＄に入れます。
４０：Ｘ＄の情報の入ったアドレスをＡに代入します。
５０：Ｘ＄の長さををＬ、Ｘ＄を格納している相対アドレスの下位バイトをＡ１、上位バイトをＡ２に代入します。
６０、７０：相対アドレスを１６進数表現に変換します。
８０：Ｘ＄の長さＬを表示します。
９０：相対アドレスを表示します。
１００：ＳＴＲＰＴＲと相対アドレスをたして、文字変数Ｘ＄を格納しているアドレスを求めます。
１１０：ＳＴＲＰＴＲの値を１６進数で表示します。
１２０：格納アドレスを１６進数で表示します。

```
X$=? ABCDEFGHIJKLMN
長さ       = 14
相対アドレス=0332
STRPTR     =672C
格納アドレス=6A5E
Ok
```

※上の実行結果は実際の結果とは異なります。

3.10.3 ＤＴＬ

機能

現在読み込み中のＤＡＴＡ文の行番号を返します。

書式

```
DTL
```

文例

ＰＲＩＮＴ　ＤＴＬ

⇨ＲＥＡＤステートメントで次に読み込むＤＡＴＡステートメントの行番号が表示されます。

解説

　ＤＡＴＡステートメントのデータを読み込んでいるとき、次に読み込まれるＤＡＴＡステートメントの行番号が、この変数の値になります。

参照

2.3.5 ＲＥＳＴＯＲＥ、2.3.3 ＲＥＡＤ

3.10

3.10.4 ERL

機能

エラー発生時に、その行番号を返します。

書式

```
ERL
```

文例

PRINT ERL

⇨エラー処理ルーチン内で値を表示すれば、エラーの発生した行番号を知ることができます。

解説

プログラムの実行中にエラーが発生した場合、「ON ERROR GOTO」ステートメントによってエラー処理ルーチンにジャンプしても、どの行でエラーが発生したのかわからないため、この変数の値を調べます。

この変数にはエラーの発生した行番号が入っているので、エラーの発生箇所を知ることができます。

ダイレクト実行により発生したエラーでは、この変数は変化しません。

参照

3.10.5 ERR、2.5.1 ON ERROR GOTO

サンプルプログラム

```
10 ' ERL (例)
20 ON ERROR GOTO 90
30 PRINT"割り算 A/B"
40 INPUT"A=";A
50 INPUT"B=";B
60 C=A/B
70 PRINT"A/B=";C
80 GOTO 40
90 BEEP
100 PRINT"ｴﾗｰｺｰﾄﾞ(ERR)=";ERR
110 PRINT"ｴﾗｰ行  (ERL)=";ERL
120 RESUME 40

RUN
割り算 A/B
A=? 2
B=? 3
A/B= .66666667
A=? 2
B=? 0
ｴﾗｰｺｰﾄﾞ(ERR)= 11
ｴﾗｰ行  (ERL)= 60
A=?
Break in 40
Ok.
```

２０：プログラムの実行中、エラーが発生すると９０番にジャンプするように設定します。

３０～８０：割り算の計算ルーチン。

９０～１２０：エラー処理ルーチン。

９０：ビープ音を鳴らします。

１００：エラーコードＥＲＲを表示します。

１１０：エラー発生行ＥＲＲを表示します。

１２０：４０番に戻ります。

プログラムを止めるには SHIFT ＋ BREAK キーを押してください。

3.10

3.10.5　ERR

機　能

エラーが発生したとき、そのエラーコードを返します。

書　式

```
ERR
```

文　例

ＩＦ　ＥＲＲ＝６　ＴＨＥＮ　ＰＲＩＮＴ"オーバーフロー"

⇨エラー処理ルーチンの中に書けば、ＥＲＲの値によっていろいろなエラーメッセージを表示することができます。上の例では、生じたエラーのコードが６のとき、「オーバーフロー」のエラーメッセージを表示します。

解　説

プログラム実行中にエラーが発生した場合、「ＯＮ　ＥＲＲＯＲ　ＧＯＴＯ」ステートメントによって、エラー処理ルーチンにジャンプしても、どんな発生したのかわからないため、この変数の値を調べます。

この変数にはエラーコードが入っているので、エラーの発生原因を知ることができます。

エラーコードについては、付録の「エラーメッセージ一覧表」を参照してください。

ダイレクト実行により発生したエラーでは、この変数は変化しません。

参　照

3.10.4 ＥＲＬ、2.5.1 ＯＮ　ＥＲＲＯＲ　ＧＯＴＯ、
付録「エラーメッセージ一覧表」

サンプルプログラム

```
10 ' ERR (例)
20 ON ERROR GOTO 90
30 PRINT"割り算 A/B"
40 INPUT"A=";A
50 INPUT"B=";B
60 C=A/B
70 PRINT"A/B=";C
80 GOTO 40
90 BEEP
100 PRINT"ｴﾗｰｺｰﾄﾞ(ERR)=";ERR
110 PRINT"ｴﾗｰ行  (ERL)=";ERL
120 RESUME 40
```

２０：プログラムの実行中、エラーが発生すると９０番にジャンプするように設定します。

３０～８０：割り算の計算ルーチン。

９０～１２０：エラー処理ルーチン。

９０：ビープ音を鳴らします。

１００：エラーコード（ＥＲＲ）を表示します。

１２０：エラー行（ＥＲＬ）を表示します。

１２０：４０番に戻ります。

3.10

```
RUN
割り算 A/B
A=? 2
B=? 3
A/B= .66666667
A=? 2
B=? 0
エラーコード(ERR)= 11
エラー行  (ERL)= 60
A=?
Break in 40
Ok.
```

プログラムを止めるには SHIFT + BREAK キーを押してください。

3.10

付　　録

A.1 コマンド・ステートメント・関数の省略形一覧表

実際プログラミングをするとき、省略形を知っていると、たいへん便利なので、下にその一覧表を示します。

（注意）　たとえば、ＬＩＳＴの省略形はＬ．ですが、実際にはそれより長いＬＩ．やＬＩＳ．も省略形として使うことができます。

そこで、下に示す省略形はいずれも省略形の中で最短のものとします。また、省略形の最後には必ずピリオド．をつけます。

省略形入力後、再び表示すると、もとの形で表示されます。

もとの形	省略形	もとの形	省略形
AKCNV$	AK.	CREV	CR.
APSS	AP.	CSIZE	CS.
ATTR$	ATT.	CSRLIN	CSR.
AUTO	A.	CSTOP	CST.
AUTO*	A.*	CURSOR	CU.
BEEP	BE.	DATA	DA.
BIN$	BI.	DEFDBL	DEFD.
BOOT	BO.	DEFINT	DEFI.
CALL	CA.	DEF KEY	DEFK.
CANVAS	CAN.	DEFSNG	DEFS.
CBLACK	CB.	DELETE	D.
CDBL	CD.	DEVICE	DEV.
CFLASH	CF.	EDIT	E.
CGEN	CG.	EJECT	EJ.
CGPAT$	CGP.	ERASE	ER.
CHAIN	CHA.	ERROR	ERR.
CHANNEL	CH.	FAST	FA.
CHARACTER$	CHAR.	FIELD	FI.
CHDIR	CHD.	FILES	FIL.
CIRCLE	CI.	FOR	F.
CIRCLE@	CI.@	FPOS	FP.
CLEAR	CLE.	FRAC	FR.
CLICK OFF	CLI. OFF	GOSUB	GOS.
CLIK ON	CLI. ON	GOTO	G.
CLOSE	CLO.	GRAPH	GR.
CMT	CM.	GRAPH@	GR.@
COLOR	COL.	HCOPY	H.
CONSOLE	CONS.	HEXCHR$	HEXC.
CONSOLE#	CONS.#	HEX$	HE.
CONT	C.	IF THEN ELSE	IF TH. EL.

もとの形	省略形
INKEY$	INK.
INPUT	I.
INPUT#	I.#
INPUT$	I.$
INSTR	INS.
JIS$	J.
KACNV$	KAC.
KANJI$	KA.
KBUF OFF	KB.OFF
KBUF ON	KB.ON
KEY	K.
KEY0	K.0
KEYLIST	K.L.
KEY OFF	K.OFF
KEY ON	K.ON
KEY STOP	K.S.
KILL	KI.
KLEN	KLE.
KLIST	KL.
KMODE	KM.
KPOS	KP.
KSEN	KS.
KTN$	KT.
LABEL	LA.
LAYER	LAY.
LEFT$	LEF.
LFILES	LF.
LIMIT	LIM.
LINE INPUT	LINEI.
LINE INPUT#	LINEI.#
LINPUT	LI.
LINPUT#	LIN.#
LIST	L.
LIST*	L.*
LIST@	L.@
LLIST	LL.
LLIST*	LL.*
LOAD	LO.
LOADM	LO.M
LOCATE	LOC.

もとの形	省略形
LPOUT	LPO.
LPRINT	LP.又はL?
LPRINT USING	LP.US.
LSET	LS.
MAXFILES	MA.
MERGE	M.
MID$	MI.
MIRROR$	MIR.
MKDIR	MK.
MKI$	MKI.
MOUSE	MOU.
MUSIC	MU.
MUSIC@	MU.@
NAME	NA.
NEXT	N.
OCT$	OC.
ON ERR GOTO	O.ERR.G.
ON KEY GOSUB	O.K.GOS.
OPTION BASE	OP.B.
OPTION SCREEN	OP.SC.
PAINT	PAI.
PAINT@	PAI.@
PALET	PAL.
PALET@	PAL.@
PATTERN	PAT.
PAUSE	PA.
PEEK	PE.
PEEK@	PE.@
PLAY	PL.
PLAY@	PL.@
POINT	POI.
POKE	PO.
POKE@	PO.@
POLY	POL.
POSITION	POS.
PRESET	PRE.
PRINT	P.又は?
PRINT USING	P.US.又は?US.
PRINT#	P.#又は?#
PRINT#0	P.#0又は?#0

もとの形	省略形
PSET	PS.
RANDOMIZE	RA.
REM	'(アポストロフィ)
RENUM	REN.
REPEAT	REP.
REPEAT OFF	REP. OFF
REPEAT ON	REP. ON
RESTORE	RES.
RESUME	RESU.
RETURN	RE.
RIGHT$	RI.
RMDIR	RM.
RSET	RS.
RUN	R.
SAVE	SA.
SAVEM	SA. M
SCREEN	SC.
SCREEN@	SC. @
SCROLL	SCRO.
SEARCH	SE.
SOUND	SO.
SOUND@	SO. @
SPACE$	SPA.
STICK	STI.
STOP	S.
STRIG	STR.
STRING$	STRIN.
STRPTR	STRP.
SWAP	SW.
SYMBOL	SY.
TEMPO	TE.
TROFF	TROF.
TRON	T.
TVPW OFF	TV. OFF
TVPW ON	TV. ON
UNTIL	U.
VARPTR	VAR.
VDIM	V.
VDIM CLEAR	V. CLE.
VERIFY	VE.

もとの形	省略形
WAIT	WA.
WEND	WE.
WHILE	W.
WIDTH	WI.
WINDOW	WIN.
WRITE	WR.
WRITE#	WR. #

〈論理演算子の省略〉

NOT=NO.　IMP=IM.
AND=AN.　EQV=EQ.
XOR=X.

A.2 コントロールコード表

制御文字は目で見ることはできませんが、そのコードは、[CTRL]（コントロールキー）を文字のキーと同時に押すことによって、キーボードから入力することができます。

このとき入力されるコードをコントロールコードといい、キャラクタコードの0～31（＝&H1F）の部分を占めています。

キーボードからコントロールコードを入力することにより、主として次のような画面の操作をすることができます。

CTRL＋Aは、[CTRL]キーを押しながら、[A]のキーを押すことを意味します。

CTRL＋	コード	処理内容
@	00	ダミー。
Aまたはa	01	インサートモード[1]にする。解除は[BREAK]で行なえる。
Bまたはb	02	現在のワード[2]の先頭にカーソルを戻す。
Cまたはc	03	実行を停止する。(＝[SHIFT]＋[BREAK])
Dまたはd	04	画面などを標準の状態に戻す。(＝INIT)
Eまたはe	05	現在のカーソル以降1行を消す。
Fまたはf	06	次のワードの先頭にカーソルを進める。
Gまたはg	07	ビープ音を鳴らす。(＝BEEP)
Hまたはh	08	1文字分消して戻る。(＝[INS DEL])
Iまたはi	09	水平タビュレーションを行う。(＝[H TAB])
Jまたはj	0A	現在のカーソル以降の次の行に分ける。
Kまたはk	0B	カーソルを画面のホーム位置に移す。(＝[CLR HOME])
Lまたはl	0C	テキスト画面を消去する。(＝[SHIFT]＋[CLR HOME])
Mまたはm	0D	キャリッジリターンをする。(＝[⏎])
Nまたはn	0E	現在のカーソルから上を上方向にスクロールする。(＝[ROLL UP])
Oまたはo	0F	現在のカーソルから下を下方向にスクロールする。(＝[ROLL DOWN])
Pまたはp	10	ダミー。
Qまたはq	11	実行の一時停止を解除する。
Rまたはr	12	空白を挿入する。(＝[SHIFT]＋[INS DEL])
Sまたはs	13	実行を一時停止する[3]（＝[BREAK])
Tまたはt	14	水平タビュレーション位置の新たな設定を行なう。
Uまたはu	15	ダミー。
Vまたはv	16	ダミー。
Wまたはw	17	現在のカーソルがあると行と次の行をつないで1つにする。
Xまたはx	18	ダミー。
Yまたはy	19	現在のカーソル位置の水平タビュレーションの解除を行なう。
Zまたはz	1A	現在のカーソルより下のテキスト画面をすべて消去する。
［	1B	ダミー。
￥	1C	カーソルを右へ移動する。(＝[⇨])
］	1D	カーソルを左へ移動する。(＝[⇦])

CTRL＋	コード	処　理　内　容
ˆ	1E	カーソルを上へ移動する。(=⇧)
—	1F	カーソルを下へ移動する。(=⇩)
0		画面の背景色を黒（透明）にする。　(=COLOR, 0)
1		画面の背景色を青にする。　(=COLOR, 1)
2		画面の背景色を赤にする。　(=COLOR, 2)
3		画面の背景色をマゼンタにする。　(=COLOR, 3)
4		画面の背景色を緑にする。　(=COLOR, 4)
5		画面の背景色をシアンにする。　(=COLOR, 5)
6		画面の背景色を黄色にする。　(=COLOR, 6)
7		画面の背景色を白にする。　(=COLOR, 7)

テンキーの　0		文字グラフィックの色を黒（透明にする)。　(=COLOR 0)
1		文字グラフィックの色を青にする。　(=COLOR 1)
2		文字グラフィックの色を赤にする。　(=COLOR 2)
3		文字グラフィックの色をマゼンタにする。　(=COLOR 3)
4		文字グラフィックの色を緑にする。　(=COLOR 4)
5		文字グラフィックの色をシアンにする。　(=COLOR 5)
6		文字グラフィックの色を黄色にする。　(=COLOR 6)
7		文字グラフィックの色を白にする。　(=COLOR 7)
／		キャラクタゼネレータのROM／RAMを切り換える。(=CGEN)
＊		文字を点滅モードとノーマルモードとに切り換える。(=CFLASH)
—		文字を反転モードとノーマルモードとに切り換える。(=CREV)

1）インサートモード（insert mode）挿入したい文字キーを押すたびに、カーソルから右の部分が自動的に右に移動して、キーボードから入力した文字が挿入されるモード。

2）ワード（word）隙間のない英数字の文字列をいいます。

3）プログラムの実行中は、キーを押している間だけ停止し、キーを離すと再び実行を開始します。

A.3　コードの体系

１バイトコード文字（半角文字）には、シフトＪＩＳコードまたはキャラクタコードが割り当てられており、２バイトコード文字（全角文字）には、シフトＪＩＳコード、区点コード、ＪＩＳ漢字コードが割り当てられています。

〔1〕キャラクタコード（ＡＳＣＩＩコード）

２５６個ある１バイトコード文字（半角文字）には、０～２５５のキャラクタコードが割り当てられています。キャラクタコードｎに対応する文字を表示するには、

　　ＰＲＩＮＴ　ＣＨＲ＄（ｎ）または　ＰＲＩＮＴ＃０，ＣＨＲ＄（ｎ）

を実行します。

ｎ＝＆Ｈ８０～＆Ｈ９Ｆ、および＆ＨＥ０～＆ＨＦＦの文字は、

　　ＫＭＯＤＥ０

を設定してからＰＲＩＮＴ　ＣＨＲ＄（ｎ）を実行しなければ表示されません。

〔2〕シフトＪＩＳコード（１６進数）

２バイトコード文字（全角文字）には、４けたの１６進数のシフトＪＩＳコードが割り当てられています。シフトＪＩＳコードｈｈｈｈに対応する文字を表示するには、

　　ＫＭＯＤＥ１

を設定してから、

　　ＰＲＩＮＴ　ＣＨＲ＄（＆Ｈｈｈｈｈ）

を実行します。

〔3〕区点コード（１０進数）

区番号（０１～９４）と点番号（０１～９４）の各２けたを区番号の順に並べて４けたで表現したコードです。ＢＡＳＩＣ中ではＪＩＳ区点定数として、先頭に＆ＫをつけてシフトＪＩＳコードの値として利用することができます。

たとえば、＆Ｋ０１０１はシフトＪＩＳコードの＆Ｈ８１４０の値をもちます。「シフトＪＩＳコードと区点コードの関係」参照。

〔4〕ＪＩＳ漢字コード（１６進数）

ＪＩＳ情報交換用漢字符号系で定められている第１バイトと第２バイトを２１～７Ｅの各２けたの１６進数で表わしたもので、各文字は４けたの１６進数で表わされます。ＢＡＳＩＣ中で用いる場合は、先頭に＆ＪをつけたＪＩＳ１６進定数にしてシフトＪＩＳコードの値として利用できます。

たとえば、＆Ｊ２１２１はシフトＪＩＳコードの＆８１４０の値をもちます。「シフトＪＩＳコードとＪＩＳ漢字コードの関係」参照。

A.3.1　1バイトコード文字（半角文字）

１バイトコード文字にはキャラクタコードの０～２５５（１６進数で０～＆ＨＦＦ）が割り振られています。

キャラクタコードに割り振られている文字には、図形文字と制御文字があります。図形文字は、画面上の表示やプリンタの印刷のための文字であり、制御文字は画面の制御とプリンタの制御のための文字です。プリンタに対する制御文字の働きについては、各プリンタの取扱説明書を参照してください。図形文字を画面やプリンタに出力するための具体的な方法は、ＰＲＩＮＴステートメントあるいはＰＲＩＮＴ＃ステートメントを参照してください。

ＫＭＯＤＥ１が実行されて、全角文字が表示されるモードになっているときは、キャラクタコードの８０～９ＦとＥ０～ＦＦがシフトＪＩＳコードの第１バイト目として使用されます。

ここでは、１バイトコード文字（２５６個）の一覧をキャラクタコードと共に載せます。

ＫＭＯＤＥ１の場合、キャラクタコード８１～９Ｆ、Ｅ０～ＦＦ（１６進数）には、２バイトコード文字の第１バイトと書かれています。これは、キャラクタコードの１２９～１５９と２２４～２５２（１０進数）の範囲は、２バイトコード文字を表わすために予約されており、この部分のキャラクタコードだけは、文字を表わさないことを示しています。

１バイトコード文字の場合（キャラクタコード表）

〔ＫＭＯＤＥの場合〕

下位ビット→

上位ビット↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	P	Q	R	S	T	U	V	W	X	Y	Z		→	←	↑	↓
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	￣	π
8																
9																
A		。	「	」	、	・	ヲ	ァ	ィ	ゥ	ェ	ォ	ャ	ュ	ョ	ッ
B	ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
C	タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ
D	ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	゛	゜
E	●	○	♠	♥	♦	♣										
F		土	金	木	水	火	月	日	時	分	秒	年	円	人	生	〒

〔ＫＭＯＤＥ１の場合〕

下位ビット→

上位ビット↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	P	Q	R	S	T	U	V	W	X	Y	Z		→	←	↑	↓
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	￣	π
8–9	２バイトコード文字の第１バイト目															
A		。	「	」	、	・	ヲ	ァ	ィ	ゥ	ェ	ォ	ャ	ュ	ョ	ッ
B	ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
C	タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ
D	ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	゛	゜
E	２バイトコード文字の第１バイト目															
F																

斜線部は未使用。ただし、次に示すコードは、ＫＥＹ０ステートメント中でコントロールコードとして使用できます。

Ｆ１～ＦＡ：　日本語入力モード中、ファンクションキーの１～１０と同じ機能を持つ。

ＦＥ　　　：　XFER のみを押すのと同じ。

ＦＦ　　　：　CTRL + XFER と同じ。

A.3.2　2バイトコード文字（全角文字）

２バイトコード文字には、シフトＪＩＳコード、区点コード、およびＪＩＳ漢字コードの３つのコードが対応しています。

シフトＪＩＳコード（１６進数）

１６進数４けたで表わしたものです。１バイトコード文字のところで述べたように、その第１バイトは、必ず８１～９Ｆ、Ｅ０～ＦＣの範囲内にあります（１バイトコード文字の一覧表で、２バイトコード文字の第１バイトとされていた部分）。さらに、第２バイトも、４０～７Ｅ、８０～ＦＣの範囲内に限られています。以下の説明で用いる図では、影の部分がこの範囲外であることを示しています。またｍｍ、ｎｎはそれぞれ、シフトＪＩＳコードの第１、第２バイトを表わしています。

シフトＪＩＳコードとＪＩＳ漢字コードの関係

mm、nnはそれぞれシフトJISコードの第1、第2バイトを示します。
▨は未使用領域です。

シフトＪＩＳコードと区点コードの関係

mm、nnはそれぞれシフトJISコードの第1、第2バイトを示します。
▨は未使用領域です。

A.3.3　非漢字およびＪＩＳ第１水準漢字一覧表

■表の見方

ＪＩＳコード、シフトＪＩＳコードは１６進数で表現されていますので、次のようにしてコードを読んでください。

(例)「移」のコードの読み方

シフトＪＩＳコード	ＪＩＳコード	区点コード	＋０	＋１	＋２	…………＋Ｃ
↓	↓	↓	↓	↓	↓	↓
８８ＣＥ	３０５０	１６４８	夷	委	威	……………移

「移」のＪＩＳコード…………「移」とＪＩＳコードとの交点の値３０５０の上位３桁３０５に、横列に割り当てられた０からＦの値のうち、Ｃを最下桁につけた３０５Ｃが「移」のＪＩＳコードです。

「移」のシフトＪＩＳコード…「移」とシフトJISコードの交点の値８８ＣＥが「夷」です。右に１文字ずつずれるごとにコードが１ずつ加算され、８８ＣＦ－委、８８Ｄ０－威、………となります。「移」のシフトＪＩＳコードは８８ＤＡとなります。

「移」の区点コード……………「移」と区点コードの交点の値１６４８が「夷」です。右に１文字ずつずれるごとにコードが１ずつ加算され、１６４９－委、１６５０－威、………となります。「移」の区点コードは１６６０となります。

非漢字

	シフトJISコード:JIS漢字コード:区点コード	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
!	813F:2120:0100:	※		、	。	，	．	・	：	；	？	！	゛	゜	´	｀	¨
	814F:2130:0116:	＾	￣	＿	ヽ	ヾ	ゝ	ゞ	〃	仝	々	〆	〇	ー	―	‐	／
	815F:2140:0132:	＼	～	∥	｜	…	‥	‘	’	“	”	（	）	〔	〕	［	］
	816F:2150:0148:	｛	｝	〈	〉	《	》	「	」	『	』	【	】	＋	－	±	×
	8180:2160:0164:	÷	＝	≠	＜	＞	≦	≧	∞	∴	♂	♀	°	′	″	℃	￥
	8190:2170:0180:	＄	￠	￡	％	＃	＆	＊	＠	§	☆	★	○	●	◎	◇	※
″	819E:2220:0200:	※	◆	□	■	△	▲	▽	▼	※	〒	→	←	↑	↓	〓	|
	81AE:2230:0216:	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|
	81BE:2240:0232:	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|
	81CE:2250:0248:	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|
	81DE:2260:0264:	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|
	81EE:2270:0280:	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|	※
＃	823F:2320:0300:	※	|	|	|	|	|	|	|	|	|	|	|	|	|	|	|
	824F:2330:0316:	０	１	２	３	４	５	６	７	８	９	|	|	|	|	|	|
	825F:2340:0332:	|	Ａ	Ｂ	Ｃ	Ｄ	Ｅ	Ｆ	Ｇ	Ｈ	Ｉ	Ｊ	Ｋ	Ｌ	Ｍ	Ｎ	Ｏ
	826F:2350:0348:	Ｐ	Ｑ	Ｒ	Ｓ	Ｔ	Ｕ	Ｖ	Ｗ	Ｘ	Ｙ	Ｚ	|	|	|	|	|
	8280:2360:0364:	|	ａ	ｂ	ｃ	ｄ	ｅ	ｆ	ｇ	ｈ	ｉ	ｊ	ｋ	ｌ	ｍ	ｎ	ｏ
	8290:2370:0380:	ｐ	ｑ	ｒ	ｓ	ｔ	ｕ	ｖ	ｗ	ｘ	ｙ	ｚ	|	|	|	|	※
＄	829E:2420:0400:	※	ぁ	あ	ぃ	い	ぅ	う	ぇ	え	ぉ	お	か	が	き	ぎ	く
	82AE:2430:0416:	ぐ	け	げ	こ	ご	さ	ざ	し	じ	す	ず	せ	ぜ	そ	ぞ	た
	82BE:2440:0432:	だ	ち	ぢ	っ	つ	づ	て	で	と	ど	な	に	ぬ	ね	の	は
	82CE:2450:0448:	ば	ぱ	ひ	び	ぴ	ふ	ぶ	ぷ	へ	べ	ぺ	ほ	ぼ	ぽ	ま	み
	82DE:2460:0464:	む	め	も	ゃ	や	ゅ	ゆ	ょ	よ	ら	り	る	れ	ろ	ゎ	わ
	82EE:2470:0480:	ゐ	ゑ	を	ん	|	|	|	|	|	|	|	|	|	|	|	※
％	833F:2520:0500:	※	ァ	ア	ィ	イ	ゥ	ウ	ェ	エ	ォ	オ	カ	ガ	キ	ギ	ク
	834F:2530:0516:	グ	ケ	ゲ	コ	ゴ	サ	ザ	シ	ジ	ス	ズ	セ	ゼ	ソ	ゾ	タ
	835F:2540:0532:	ダ	チ	ヂ	ッ	ツ	ヅ	テ	デ	ト	ド	ナ	ニ	ヌ	ネ	ノ	ハ

<table>
<tr><td></td><td>シフトJISコード:JIS漢字コード:区点コード</td><td>+0</td><td>+1</td><td>+2</td><td>+3</td><td>+4</td><td>+5</td><td>+6</td><td>+7</td><td>+8</td><td>+9</td><td>+A</td><td>+B</td><td>+C</td><td>+D</td><td>+E</td><td>+F</td></tr>
<tr><td></td><td>836F:2550:0548:</td><td>パ</td><td>パ</td><td>ヒ</td><td>ビ</td><td>ピ</td><td>フ</td><td>ブ</td><td>プ</td><td>ヘ</td><td>ベ</td><td>ペ</td><td>ホ</td><td>ボ</td><td>ポ</td><td>マ</td><td>ミ</td></tr>
<tr><td></td><td>8380:2560:0564:</td><td>ム</td><td>メ</td><td>モ</td><td>ャ</td><td>ヤ</td><td>ュ</td><td>ユ</td><td>ョ</td><td>ヨ</td><td>ラ</td><td>リ</td><td>ル</td><td>レ</td><td>ロ</td><td>ヮ</td><td>ワ</td></tr>
<tr><td></td><td>8390:2570:0580:</td><td>キ</td><td>ヱ</td><td>ヲ</td><td>ン</td><td>ヴ</td><td>ヵ</td><td>ヶ</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>※</td></tr>
<tr><td>&</td><td>839E:2620:0600:</td><td>※</td><td>Α</td><td>Β</td><td>Γ</td><td>Δ</td><td>Ε</td><td>Ζ</td><td>Η</td><td>Θ</td><td>Ι</td><td>Κ</td><td>Λ</td><td>Μ</td><td>Ν</td><td>Ξ</td><td>Ο</td></tr>
<tr><td></td><td>83AE:2630:0616:</td><td>Π</td><td>Ρ</td><td>Σ</td><td>Τ</td><td>Υ</td><td>Φ</td><td>Χ</td><td>Ψ</td><td>Ω</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td></tr>
<tr><td></td><td>83BE:2640:0632:</td><td>|</td><td>α</td><td>β</td><td>γ</td><td>δ</td><td>ε</td><td>ζ</td><td>η</td><td>θ</td><td>ι</td><td>κ</td><td>λ</td><td>μ</td><td>ν</td><td>ξ</td><td>ο</td></tr>
<tr><td></td><td>83CE:2650:0648:</td><td>π</td><td>ρ</td><td>σ</td><td>τ</td><td>υ</td><td>φ</td><td>χ</td><td>ψ</td><td>ω</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td></tr>
<tr><td></td><td>83DE:2660:0664:</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td></tr>
<tr><td></td><td>83EE:2670:0680:</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>※</td></tr>
<tr><td>′</td><td>843F:2720:0700:</td><td>※</td><td>А</td><td>Б</td><td>В</td><td>Г</td><td>Д</td><td>Е</td><td>Ё</td><td>Ж</td><td>З</td><td>И</td><td>Й</td><td>К</td><td>Л</td><td>М</td><td>Н</td></tr>
<tr><td></td><td>844F:2730:0716:</td><td>О</td><td>П</td><td>Р</td><td>С</td><td>Т</td><td>У</td><td>Ф</td><td>Х</td><td>Ц</td><td>Ч</td><td>Ш</td><td>Щ</td><td>Ъ</td><td>Ы</td><td>Ь</td><td>Э</td></tr>
<tr><td></td><td>845F:2740:0732:</td><td>Ю</td><td>Я</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td></tr>
<tr><td></td><td>846F:2750:0748:</td><td>|</td><td>а</td><td>б</td><td>в</td><td>г</td><td>д</td><td>е</td><td>ё</td><td>ж</td><td>з</td><td>и</td><td>й</td><td>к</td><td>л</td><td>м</td><td>н</td></tr>
<tr><td></td><td>8480:2760:0764:</td><td>о</td><td>п</td><td>р</td><td>с</td><td>т</td><td>у</td><td>ф</td><td>х</td><td>ц</td><td>ч</td><td>ш</td><td>щ</td><td>ъ</td><td>ы</td><td>ь</td><td>э</td></tr>
<tr><td></td><td>8490:2770:0780:</td><td>ю</td><td>я</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>※</td></tr>
</table>

漢字第一水準

<table>
<tr><td>0</td><td>889E:3020:1600:</td><td>※</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>ア</td><td>889E:3020:1600:</td><td></td><td>亜</td><td>唖</td><td>娃</td><td>阿</td><td>哀</td><td>愛</td><td>挨</td><td>姶</td><td>逢</td><td>葵</td><td>茜</td><td>穐</td><td>悪</td><td>握</td><td>渥</td></tr>
<tr><td></td><td>88AE:3030:1616:</td><td>旭</td><td>葦</td><td>芦</td><td>鯵</td><td>梓</td><td>圧</td><td>斡</td><td>扱</td><td>宛</td><td>姐</td><td>虻</td><td>飴</td><td>絢</td><td>綾</td><td>鮎</td><td>或</td></tr>
<tr><td></td><td>88BE:3040:1632:</td><td>粟</td><td>袷</td><td>安</td><td>庵</td><td>按</td><td>暗</td><td>案</td><td>闇</td><td>鞍</td><td>杏</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>イ</td><td>88BE:3040:1632:</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>以</td><td>伊</td><td>位</td><td>依</td><td>偉</td><td>囲</td></tr>
<tr><td></td><td>88CE:3050:1648:</td><td>夷</td><td>委</td><td>威</td><td>尉</td><td>惟</td><td>意</td><td>慰</td><td>易</td><td>椅</td><td>為</td><td>畏</td><td>異</td><td>移</td><td>維</td><td>緯</td><td>胃</td></tr>
<tr><td></td><td>88DE:3060:1664:</td><td>萎</td><td>衣</td><td>謂</td><td>違</td><td>遺</td><td>医</td><td>井</td><td>亥</td><td>域</td><td>育</td><td>郁</td><td>磯</td><td>一</td><td>壱</td><td>溢</td><td>逸</td></tr>
<tr><td></td><td>88EE:3070:1680:</td><td>稲</td><td>茨</td><td>芋</td><td>鰯</td><td>允</td><td>印</td><td>咽</td><td>員</td><td>因</td><td>姻</td><td>引</td><td>飲</td><td>淫</td><td>胤</td><td>蔭</td><td>※</td></tr>
<tr><td>1</td><td>893F:3120:1700:</td><td>※</td><td>院</td><td>陰</td><td>隠</td><td>韻</td><td>吋</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>ウ</td><td>893F:3120:1700:</td><td></td><td></td><td></td><td></td><td></td><td></td><td>右</td><td>宇</td><td>烏</td><td>羽</td><td>迂</td><td>雨</td><td>卯</td><td>鵜</td><td>窺</td><td>丑</td></tr>
<tr><td></td><td>894F:3130:1716:</td><td>碓</td><td>臼</td><td>渦</td><td>嘘</td><td>唄</td><td>欝</td><td>蔚</td><td>鰻</td><td>姥</td><td>厩</td><td>浦</td><td>瓜</td><td>閏</td><td>噂</td><td>云</td><td>運</td></tr>
<tr><td></td><td>895F:3140:1732:</td><td>雲</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>エ</td><td>895F:3140:1732:</td><td></td><td>荏</td><td>餌</td><td>叡</td><td>営</td><td>嬰</td><td>影</td><td>映</td><td>曳</td><td>栄</td><td>永</td><td>泳</td><td>洩</td><td>瑛</td><td>盈</td><td>穎</td></tr>
<tr><td></td><td>896F:3150:1748:</td><td>頴</td><td>英</td><td>衛</td><td>詠</td><td>鋭</td><td>液</td><td>疫</td><td>益</td><td>駅</td><td>悦</td><td>謁</td><td>越</td><td>閲</td><td>榎</td><td>厭</td><td>円</td></tr>
<tr><td></td><td>8980:3160:1764:</td><td>園</td><td>堰</td><td>奄</td><td>宴</td><td>延</td><td>怨</td><td>掩</td><td>援</td><td>沿</td><td>演</td><td>炎</td><td>焔</td><td>煙</td><td>燕</td><td>猿</td><td>縁</td></tr>
<tr><td></td><td>8990:3170:1780:</td><td>艶</td><td>苑</td><td>薗</td><td>遠</td><td>鉛</td><td>鴛</td><td>塩</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>オ</td><td>8990:3170:1780:</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>於</td><td>汚</td><td>甥</td><td>凹</td><td>央</td><td>奥</td><td>往</td><td>応</td><td>※</td></tr>
<tr><td>2</td><td>899E:3220:1800:</td><td>※</td><td>押</td><td>旺</td><td>横</td><td>欧</td><td>殴</td><td>王</td><td>翁</td><td>襖</td><td>鴬</td><td>鴎</td><td>黄</td><td>岡</td><td>沖</td><td>荻</td><td>億</td></tr>
<tr><td></td><td>89AE:3230:1816:</td><td>屋</td><td>憶</td><td>臆</td><td>桶</td><td>牡</td><td>乙</td><td>俺</td><td>卸</td><td>恩</td><td>温</td><td>穏</td><td>音</td><td></td><td></td><td></td><td></td></tr>
<tr><td>カ</td><td>89AE:3230:1816:</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>下</td><td>化</td><td>仮</td><td>何</td></tr>
<tr><td></td><td>89BE:3240:1832:</td><td>伽</td><td>価</td><td>佳</td><td>加</td><td>可</td><td>嘉</td><td>夏</td><td>嫁</td><td>家</td><td>寡</td><td>科</td><td>暇</td><td>果</td><td>架</td><td>歌</td><td>河</td></tr>
<tr><td></td><td>89CE:3250:1848:</td><td>火</td><td>珂</td><td>禍</td><td>禾</td><td>稼</td><td>箇</td><td>花</td><td>苛</td><td>茄</td><td>荷</td><td>華</td><td>菓</td><td>蝦</td><td>課</td><td>嘩</td><td>貨</td></tr>
<tr><td></td><td>89DE:3260:1864:</td><td>迦</td><td>過</td><td>霞</td><td>蚊</td><td>俄</td><td>峨</td><td>我</td><td>牙</td><td>画</td><td>臥</td><td>芽</td><td>蛾</td><td>賀</td><td>雅</td><td>餓</td><td>駕</td></tr>
<tr><td></td><td>89EE:3270:1880:</td><td>介</td><td>会</td><td>解</td><td>回</td><td>塊</td><td>壊</td><td>廻</td><td>快</td><td>怪</td><td>悔</td><td>恢</td><td>懐</td><td>戒</td><td>拐</td><td>改</td><td>※</td></tr>
<tr><td>3</td><td>8A3F:3320:1900:</td><td>※</td><td>魁</td><td>晦</td><td>械</td><td>海</td><td>灰</td><td>界</td><td>皆</td><td>絵</td><td>芥</td><td>蟹</td><td>開</td><td>階</td><td>貝</td><td>凱</td><td>劾</td></tr>
<tr><td></td><td>8A4F:3330:1916:</td><td>外</td><td>咳</td><td>害</td><td>崖</td><td>慨</td><td>概</td><td>涯</td><td>碍</td><td>蓋</td><td>街</td><td>該</td><td>鎧</td><td>骸</td><td>浬</td><td>馨</td><td>蛙</td></tr>
<tr><td></td><td>8A5F:3340:1932:</td><td>垣</td><td>柿</td><td>蛎</td><td>鈎</td><td>劃</td><td>嚇</td><td>各</td><td>廓</td><td>拡</td><td>撹</td><td>格</td><td>核</td><td>殻</td><td>獲</td><td>確</td><td>穫</td></tr>
<tr><td></td><td>8A6F:3350:1948:</td><td>覚</td><td>角</td><td>赫</td><td>較</td><td>郭</td><td>閣</td><td>隔</td><td>革</td><td>学</td><td>岳</td><td>楽</td><td>額</td><td>顎</td><td>掛</td><td>笠</td><td>樫</td></tr>
<tr><td></td><td>8A80:3360:1964:</td><td>橿</td><td>梶</td><td>鰍</td><td>潟</td><td>割</td><td>喝</td><td>恰</td><td>括</td><td>活</td><td>渇</td><td>滑</td><td>葛</td><td>褐</td><td>轄</td><td>且</td><td>鰹</td></tr>
</table>

	シフト JIS コード:JIS 漢字 コード:区点 コード	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
	8A90:3370:1980:	叶	椛	樺	鞄	株	兜	竈	蒲	釜	鎌	嚙	鴨	栢	茅	萱	※
4	8A9E:3420:2000:	※	粥	刈	苅	瓦	乾	侃	冠	寒	刊	勘	勧	巻	喚	堪	姦
	8AAE:3430:2016:	完	官	寛	干	幹	患	感	慣	憾	換	敢	柑	桓	棺	款	歓
	8ABE:3440:2032:	汗	漢	澗	灌	環	甘	監	看	竿	管	簡	緩	缶	翰	肝	艦
	8ACE:3450:2048:	莞	観	諫	貫	還	鑑	間	閑	関	陥	韓	館	舘	丸	含	岸
	8ADE:3460:2064:	巌	玩	癌	眼	岩	翫	贋	雁	頑	顔	願					
キ	8ADE:3460:2064:												企	伎	危	喜	器
	8AEE:3470:2080:	基	奇	嬉	寄	岐	希	幾	忌	揮	机	旗	既	期	棋	棄	※
5	8B3F:3520:2100:	※	機	帰	毅	気	汽	畿	祈	季	稀	紀	徽	規	記	貴	起
	8B4F:3530:2116:	軌	輝	飢	騎	鬼	亀	偽	儀	妓	宜	戯	技	擬	欺	犠	疑
	8B5F:3540:2132:	祇	義	蟻	誼	議	掬	菊	鞠	吉	吃	喫	桔	橘	詰	砧	杵
	8B6F:3550:2148:	黍	却	客	脚	虐	逆	丘	久	仇	休	及	吸	宮	弓	急	救
	8B80:3560:2164:	朽	求	汲	泣	灸	球	究	窮	笈	級	糾	給	旧	牛	去	居
	8B90:3570:2180:	巨	拒	拠	挙	渠	虚	許	距	鋸	漁	禦	魚	亨	享	京	※
6	8B9E:3620:2200:	※	供	侠	僑	兇	競	共	凶	協	匡	卿	叫	喬	境	峡	強
	8BAE:3630:2216:	彊	怯	恐	恭	挟	教	橋	況	狂	狭	矯	胸	脅	興	蕎	郷
	8BBE:3640:2232:	鏡	響	饗	驚	仰	凝	堯	暁	業	局	曲	極	玉	桐	粁	僅
	8BCE:3650:2248:	勤	均	巾	錦	斤	欣	欽	琴	禁	禽	筋	緊	芹	菌	衿	襟
	8BDE:3660:2264:	謹	近	金	吟	銀											
ク	8BDE:3660:2264:						九	倶	句	区	狗	玖	矩	苦	躯	駆	駈
	8BEE:3670:2280:	駒	具	愚	虞	喰	空	偶	寓	遇	隅	串	櫛	釧	屑	屈	※
7	8C3F:3720:2300:	※	掘	窟	沓	靴	轡	窪	熊	隈	粂	栗	繰	桑	鍬	勲	君
	8C4F:3730:2316:	薫	訓	群	軍	郡											
ケ	8C4F:3730:2316:						卦	袈	祁	係	傾	刑	兄	啓	圭	珪	型
	8C5F:3740:2332:	契	形	径	恵	慶	慧	憩	掲	携	敬	景	桂	渓	畦	稽	系
	8C6F:3750:2348:	経	継	繋	罫	茎	荊	蛍	計	詣	警	軽	頸	鶏	芸	迎	鯨
	8C80:3760:2364:	劇	戟	撃	激	隙	桁	傑	欠	決	潔	穴	結	血	訣	月	件
	8C90:3770:2380:	倹	倦	健	兼	券	剣	喧	圏	堅	嫌	建	憲	懸	拳	捲	※
8	8C9E:3820:2400:	※	検	権	牽	犬	献	研	硯	絹	県	肩	見	謙	賢	軒	遣
	8CAE:3830:2416:	鍵	険	顕	験	鹼	元	原	厳	幻	弦	減	源	玄	現	絃	舷
	8CBE:3840:2432:	言	諺	限													
コ	8CBE:3840:2432:				乎	個	古	呼	固	姑	孤	己	庫	弧	戸	故	枯
	8CCE:3850:2448:	湖	狐	糊	袴	股	胡	菰	虎	誇	跨	鈷	雇	顧	鼓	五	互
	8CDE:3860:2464:	伍	午	呉	吾	娯	後	御	悟	梧	檎	瑚	碁	語	誤	護	醐
	8CEE:3870:2480:	乞	鯉	交	佼	侯	候	倖	光	公	功	効	勾	厚	口	向	※
9	8D3F:3920:2500:	※	后	喉	坑	垢	好	孔	孝	宏	工	巧	巷	幸	広	庚	康
	8D4F:3930:2516:	弘	恒	慌	抗	拘	控	攻	昂	晃	更	杭	校	梗	構	江	洪
	8D5F:3940:2532:	浩	港	溝	甲	皇	硬	稿	糠	紅	紘	絞	綱	耕	考	肯	肱
	8D6F:3950:2548:	腔	膏	航	荒	行	衡	講	貢	購	郊	酵	鉱	礦	鋼	閤	降
	8D80:3960:2564:	項	香	高	鴻	剛	劫	号	合	壕	拷	濠	豪	轟	麴	克	刻
	8D90:3970:2580:	告	国	穀	酷	鵠	黒	獄	漉	腰	甑	忽	惚	骨	狛	込	※
:	8D9E:3A20:2600:	※	此	頃	今	困	坤	墾	婚	恨	懇	昏	昆	根	梱	混	痕
	8DAE:3A30:2616:	紺	艮	魂													
サ	8DAE:3A30:2616:				些	佐	叉	唆	嵯	左	差	査	沙	瑳	砂	詐	鎖
	8DBE:3A40:2632:	裟	坐	座	挫	債	催	再	最	哉	塞	妻	宰	彩	才	採	栽
	8DCE:3A50:2648:	歳	済	災	采	犀	砕	砦	祭	斎	細	菜	裁	載	際	剤	在
	8DDE:3A60:2664:	材	罪	財	冴	坂	阪	堺	榊	肴	咲	崎	埼	碕	鷺	作	削
	8DEE:3A70:2680:	咋	搾	昨	朔	柵	窄	策	索	錯	桜	鮭	笹	匙	冊	刷	※

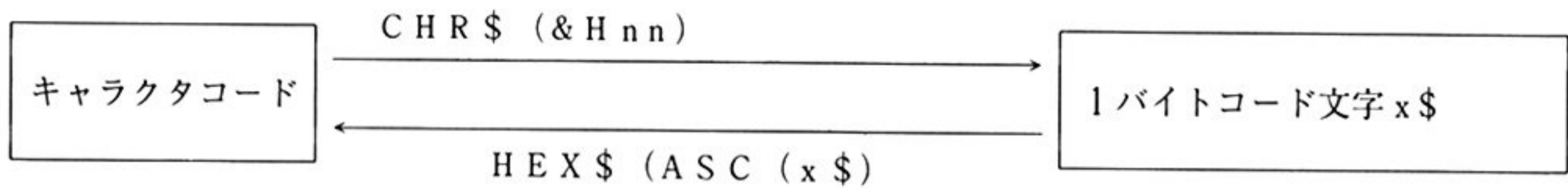
	シフト JIS コード	JIS 漢字 コード	区点 コード	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
;	8E3F	3B20	2700	※	察	拶	撮	擦	札	殺	薩	雑	皐	鯖	捌	錆	鮫	皿	晒
	8E4F	3B30	2716	三	傘	参	山	惨	撒	散	桟	燦	珊	産	算	纂	蚕	讃	賛
	8E5F	3B40	2732	酸	餐	斬	暫	残											
シ	8E5F	3B40	2732						仕	仔	伺	使	刺	司	史	嗣	四	士	始
	8E6F	3B50	2748	姉	姿	子	屍	市	師	志	思	指	支	孜	斯	施	旨	枝	止
	8E80	3B60	2764	死	氏	獅	祉	私	糸	紙	紫	肢	脂	至	視	詞	詩	試	誌
	8E90	3B70	2780	諮	資	賜	雌	飼	歯	事	似	侍	児	字	寺	慈	持	時	※
<	8E9E	3C20	2800	※	次	滋	治	爾	璽	痔	磁	示	而	耳	自	蒔	辞	汐	鹿
	8EAE	3C30	2816	式	識	鴫	竺	軸	宍	雫	七	叱	執	失	嫉	室	悉	湿	漆
	8EBE	3C40	2832	疾	質	実	蔀	篠	偲	柴	芝	屡	蕊	縞	舎	写	射	捨	赦
	8ECE	3C50	2848	斜	煮	社	紗	者	謝	車	遮	蛇	邪	借	勺	尺	杓	灼	爵
	8EDE	3C60	2864	酌	釈	錫	若	寂	弱	惹	主	取	守	手	朱	殊	狩	珠	種
	8EEE	3C70	2880	腫	趣	酒	首	儒	受	呪	寿	授	樹	綬	需	囚	収	周	※
=	8F3F	3D20	2900	※	宗	就	州	修	愁	拾	洲	秀	秋	終	繍	習	臭	舟	蒐
	8F4F	3D30	2916	衆	襲	讐	蹴	輯	週	酋	酬	集	醜	什	住	充	十	従	戎
	8F5F	3D40	2932	柔	汁	渋	獣	縦	重	銃	叔	夙	宿	淑	祝	縮	粛	塾	熟
	8F6F	3D50	2948	出	術	述	俊	峻	春	瞬	竣	舜	駿	准	循	旬	楯	殉	淳
	8F80	3D60	2964	準	潤	盾	純	巡	遵	醇	順	処	初	所	暑	曙	渚	庶	緒
	8F90	3D70	2980	署	書	薯	藷	諸	助	叙	女	序	徐	恕	鋤	除	傷	償	※
>	8F9E	3E20	3000	※	勝	匠	升	召	哨	商	唱	嘗	奨	妾	娼	宵	将	小	少
	8FAE	3E30	3016	尚	庄	床	廠	彰	承	抄	招	掌	捷	昇	昌	昭	晶	松	梢
	8FBE	3E40	3032	樟	樵	沼	消	渉	湘	焼	焦	照	症	省	硝	礁	祥	称	章
	8FCE	3E50	3048	笑	粧	紹	肖	菖	蒋	蕉	衝	裳	訟	証	詔	詳	象	賞	醤
	8FDE	3E60	3064	鉦	鍾	鐘	障	鞘	上	丈	丞	乗	冗	剰	城	場	壌	嬢	常
	8FEE	3E70	3080	情	擾	条	杖	浄	状	畳	穣	蒸	譲	醸	錠	嘱	埴	飾	※
?	903F	3F20	3100	※	拭	植	殖	燭	織	職	色	触	食	蝕	辱	尻	伸	信	侵
	904F	3F30	3116	唇	娠	寝	審	心	慎	振	新	晋	森	榛	浸	深	申	疹	真
	905F	3F40	3132	神	秦	紳	臣	芯	薪	親	診	身	辛	進	針	震	人	仁	刃
	906F	3F50	3148	塵	壬	尋	甚	尽	腎	訊	迅	陣	靭						
ス	906F	3F50	3148											笥	諏	須	酢	図	厨
	9080	3F60	3164	逗	吹	垂	帥	推	水	炊	睡	粋	翠	衰	遂	酔	錐	錘	随
	9090	3F70	3180	瑞	髄	崇	嵩	数	枢	趨	雛	据	杉	椙	菅	頗	雀	裾	※
@	909E	4020	3200	※	澄	摺	寸												
セ	909E	4020	3200					世	瀬	畝	是	凄	制	勢	姓	征	性	成	政
	90AE	4030	3216	整	星	晴	棲	栖	正	清	牲	生	盛	精	聖	声	製	西	誠
	90BE	4040	3232	誓	請	逝	醒	青	静	斉	税	脆	隻	席	惜	戚	斥	昔	析
	90CE	4050	3248	石	積	籍	績	脊	責	赤	跡	蹟	碩	切	拙	接	摂	折	設
	90DE	4060	3264	窃	節	説	雪	絶	舌	蝉	仙	先	千	占	宣	専	尖	川	戦
	90EE	4070	3280	扇	撰	栓	栴	泉	浅	洗	染	潜	煎	煽	旋	穿	箭	線	※
A	913F	4120	3300	※	繊	羨	腺	舛	船	薦	詮	賎	践	選	遷	銭	銑	閃	鮮
	914F	4130	3316	前	善	漸	然	全	禅	繕	膳	糎							
ソ	914F	4130	3316										噌	塑	岨	措	曾	曽	楚
	915F	4140	3332	狙	疏	疎	礎	祖	租	粗	素	組	蘇	訴	阻	遡	鼠	僧	創
	916F	4150	3348	双	叢	倉	喪	壮	奏	爽	宋	層	匝	惣	想	捜	掃	挿	掻
	9180	4160	3364	操	早	曹	巣	槍	槽	漕	燥	争	痩	相	窓	糟	総	綜	聡
	9190	4170	3380	草	荘	葬	蒼	藻	装	走	送	遭	鎗	霜	騒	像	増	憎	※
B	919E	4220	3400	※	臓	蔵	贈	造	促	側	則	即	息	捉	束	測	足	速	俗
	91AE	4230	3416	属	賊	族	続	卒	袖	其	揃	存	孫	尊	損	村	遜		

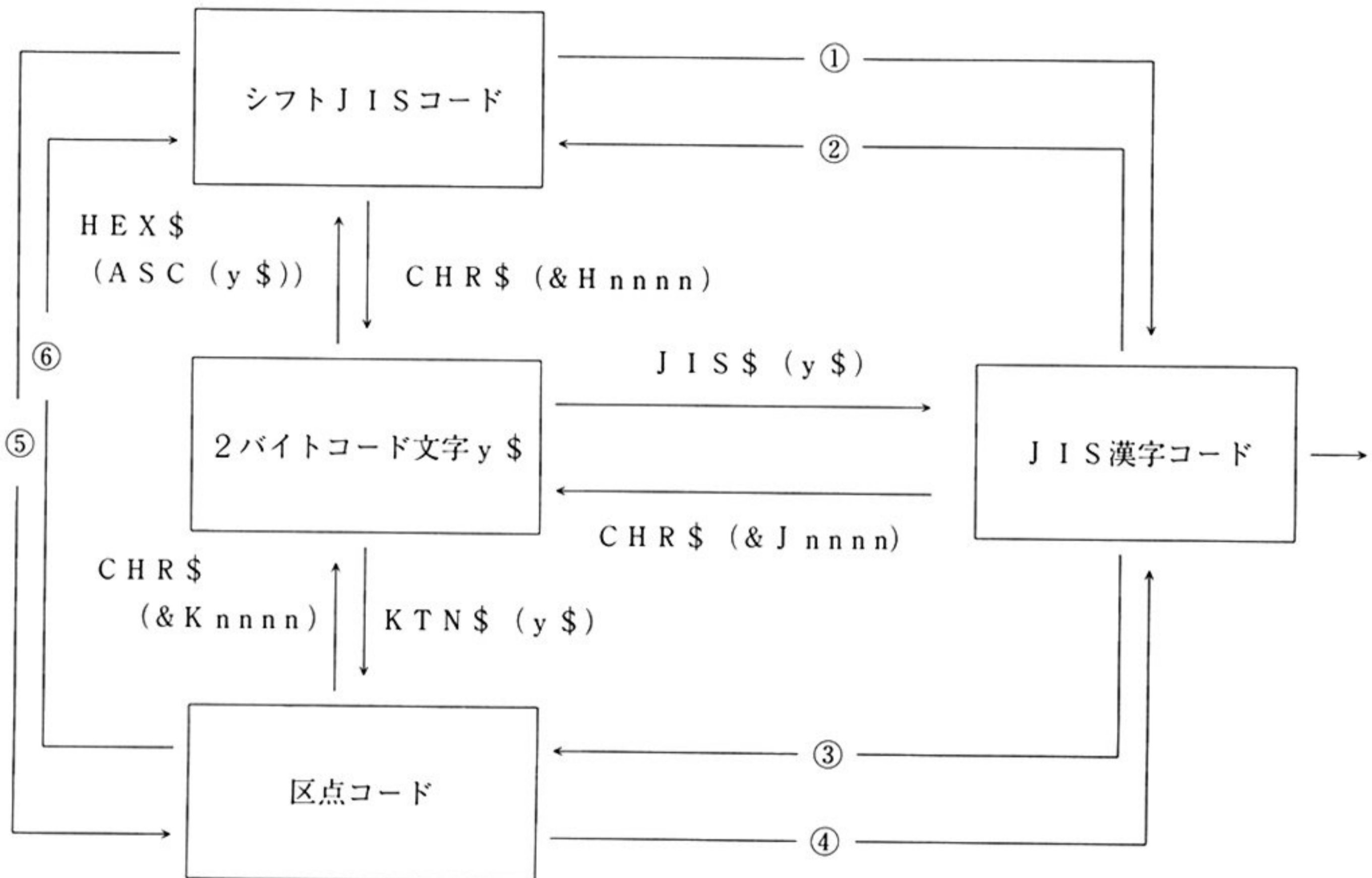
		シフト JIS コード	JIS 漢字 コード	区点 コード	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
タ		91AE	4230	3416															他	多
		91BE	4240	3432	太	汰	詑	唾	堕	妥	惰	打	柁	舵	楕	陀	駄	騨	体	堆
		91CE	4250	3448	対	耐	岱	帯	待	怠	態	戴	替	泰	滞	胎	腿	苔	袋	貸
		91DE	4260	3464	退	逮	隊	黛	鯛	代	台	大	第	醍	題	鷹	滝	瀧	卓	啄
		91EE	4270	3480	宅	托	択	拓	沢	濯	琢	託	鐸	濁	諾	茸	凧	蛸	只	※
	C	923F	4320	3500	※	叩	但	達	辰	奪	脱	巽	竪	辿	棚	谷	狸	鱈	樽	誰
		924F	4330	3516	丹	単	嘆	坦	担	探	旦	歎	淡	湛	炭	短	端	箪	綻	耽
		925F	4340	3532	胆	蛋	誕	鍛	団	壇	弾	断	暖	檀	段	男	談			
チ		925F	4340	3532														値	知	地
		926F	4350	3548	弛	恥	智	池	痴	稚	置	致	蜘	遅	馳	築	畜	竹	筑	蓄
		9280	4360	3564	逐	秩	窒	茶	嫡	着	中	仲	宙	忠	抽	昼	柱	注	虫	衷
		9290	4370	3580	註	酎	鋳	駐	樗	瀦	猪	苧	著	貯	丁	兆	凋	喋	寵	※
	D	929E	4420	3600	※	帖	帳	庁	弔	張	彫	徴	懲	挑	暢	朝	潮	牒	町	眺
		92AE	4430	3616	聴	脹	腸	蝶	調	諜	超	跳	銚	長	頂	鳥	勅	捗	直	朕
		92BE	4440	3632	沈	珍	賃	鎮	陳											
ツ		92BE	4440	3632						津	墜	椎	槌	追	鎚	痛	通	塚	栂	掴
		92CE	4450	3648	槻	佃	漬	柘	辻	蔦	綴	鍔	椿	潰	坪	壷	嬬	紬	爪	吊
		92DE	4460	3664	釣	鶴														
テ		92DE	4460	3664			亭	低	停	偵	剃	貞	呈	堤	定	帝	底	庭	廷	弟
		92EE	4470	3680	悌	抵	挺	提	梯	汀	碇	禎	程	締	艇	訂	諦	蹄	逓	※
	E	933F	4520	3700	※	邸	鄭	釘	鼎	泥	摘	擢	敵	滴	的	笛	適	鏑	溺	哲
		934F	4530	3716	徹	撤	轍	迭	鉄	典	填	天	展	店	添	纏	甜	貼	転	顛
		935F	4540	3732	点	伝	殿	澱	田	電										
ト		935F	4540	3732							兎	吐	堵	塗	妬	屠	徒	斗	杜	渡
		936F	4550	3748	登	菟	賭	途	都	鍍	砥	礪	努	度	土	奴	怒	倒	党	冬
		9380	4560	3764	凍	刀	唐	塔	塘	套	宕	島	嶋	悼	投	搭	東	桃	梼	棟
		9390	4570	3780	盗	淘	湯	涛	灯	燈	当	痘	祷	等	答	筒	糖	統	到	※
	F	939E	4620	3800	※	董	蕩	藤	討	謄	豆	踏	逃	透	鐙	陶	頭	騰	闘	働
		93AE	4630	3816	動	同	堂	導	憧	撞	洞	瞳	童	胴	萄	道	銅	峠	鴇	匿
		93BE	4640	3832	得	徳	涜	特	督	禿	篤	毒	独	読	栃	橡	凸	突	椴	届
		93CE	4650	3848	鳶	苫	寅	酉	瀞	噸	屯	惇	敦	沌	豚	遁	頓	呑	曇	鈍
ナ		93DE	4660	3864	奈	那	内	乍	凪	薙	謎	灘	捺	鍋	楢	馴	縄	畷	南	楠
		93EE	4670	3880	軟	難	汝													
ニ		93EE	4670	3880				二	尼	弐	迩	匂	賑	肉	虹	廿	日	乳	入	※
	G	943F	4720	3900	※	如	尿	韮	任	妊	忍	認								
ヌ		943F	4720	3900									濡							
ネ		943F	4720	3900										禰	祢	寧	葱	猫	熱	年
		944F	4730	3916	念	捻	撚	燃	粘											
ノ		944F	4730	3916						乃	廼	之	埜	嚢	悩	濃	納	能	脳	膿
		945F	4740	3932	農	覗	蚤													
ハ		945F	4740	3932				巴	把	播	覇	杷	波	派	琶	破	婆	罵	芭	馬
		946F	4750	3948	俳	廃	拝	排	敗	杯	盃	牌	背	肺	輩	配	倍	培	媒	梅

		シフト JIS コード : JIS 漢字コード : 区点コード	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
		9480:4760:3964:	楳	煤	狽	買	売	賠	陪	這	蝿	秤	矧	萩	伯	剥	博	拍
		9490:4770:3980:	柏	泊	白	箔	粕	舶	薄	迫	曝	漠	爆	縛	莫	駁	麦	※
	H	949E:4820:4000:	※	函	箱	硲	箸	肇	筈	櫨	幡	肌	畑	畠	八	鉢	溌	発
		94AE:4830:4016:	醗	髪	伐	罰	抜	筏	閥	鳩	噺	塙	蛤	隼	伴	判	半	反
		94BE:4840:4032:	叛	帆	搬	斑	板	氾	汎	版	犯	班	畔	繁	般	藩	販	範
		94CE:4850:4048:	釆	煩	頒	飯	挽	晩	番	盤	磐	蕃	蛮					
ヒ		94CE:4850:4048:												匪	卑	否	妃	庇
		94DE:4860:4064:	彼	悲	扉	批	披	斐	比	泌	疲	皮	碑	秘	緋	罷	肥	被
		94EE:4870:4080:	誹	費	避	非	飛	樋	簸	備	尾	微	枇	毘	琵	眉	美	※
	I	953F:4920:4100:	※	鼻	柊	稗	匹	疋	髭	彦	膝	菱	肘	弼	必	畢	筆	逼
		954F:4930:4116:	桧	姫	媛	紐	百	謬	俵	彪	標	氷	漂	瓢	票	表	評	豹
		955F:4940:4132:	廟	描	病	秒	苗	錨	鋲	蒜	蛭	鰭	品	彬	斌	浜	瀕	貧
		956F:4950:4148:	賓	頻	敏	瓶												
フ		956F:4950:4148:					不	付	埠	夫	婦	富	冨	布	府	怖	扶	敷
		9580:4960:4164:	斧	普	浮	父	符	腐	膚	芙	譜	負	賦	赴	阜	附	侮	撫
		9590:4970:4180:	武	舞	葡	蕪	部	封	楓	風	葺	蕗	伏	副	復	幅	服	※
	J	959E:4A20:4200:	※	福	腹	複	覆	淵	弗	払	沸	仏	物	鮒	分	吻	噴	墳
		95AE:4A30:4216:	憤	扮	焚	奮	粉	糞	紛	雰	文	聞						
ヘ		95AE:4A30:4216:											丙	併	兵	塀	幣	平
		95BE:4A40:4232:	弊	柄	並	蔽	閉	陛	米	頁	僻	壁	癖	碧	別	瞥	蔑	箆
		95CE:4A50:4248:	偏	変	片	篇	編	辺	返	遍	便	勉	娩	弁	鞭			
ホ		95CE:4A50:4248:														保	舗	鋪
		95DE:4A60:4264:	圃	捕	歩	甫	補	輔	穂	募	墓	慕	戊	暮	母	簿	菩	倣
		95EE:4A70:4280:	俸	包	呆	報	奉	宝	峰	峯	崩	庖	抱	捧	放	方	朋	※
	K	963F:4B20:4300:	※	法	泡	烹	砲	縫	胞	芳	萌	蓬	蜂	褒	訪	豊	邦	鋒
		964F:4B30:4316:	飽	鳳	鵬	乏	亡	傍	剖	坊	妨	帽	忘	忙	房	暴	望	某
		965F:4B40:4332:	棒	冒	紡	肪	膨	謀	貌	貿	鉾	防	吠	頬	北	僕	卜	墨
		966F:4B50:4348:	撲	朴	牧	睦	穆	釦	勃	没	殆	堀	幌	奔	本	翻	凡	盆
マ		9680:4B60:4364:	摩	磨	魔	麻	埋	妹	昧	枚	毎	哩	槙	幕	膜	枕	鮪	柾
		9690:4B70:4380:	鱒	桝	亦	俣	又	抹	末	沫	迄	侭	繭	麿	万	慢	満	※
	L	969E:4C20:4400:	※	漫	蔓													
ミ		969E:4C20:4400:				味	未	魅	巳	箕	岬	密	蜜	湊	蓑	稔	脈	妙
		96AE:4C30:4416:	粍	民	眠													
ム		96AE:4C30:4416:				務	夢	無	牟	矛	霧	鵡	椋	婿	娘			
メ		96AE:4C30:4416:														冥	名	命
		96BE:4C40:4432:	明	盟	迷	銘	鳴	姪	牝	滅	免	棉	綿	緬	面	麺		
モ		96BE:4C40:4432:															摸	模
		96CE:4C50:4448:	茂	妄	孟	毛	猛	盲	網	耗	蒙	儲	木	黙	目	杢	勿	餅
		96DE:4C60:4464:	尤	戻	籾	貰	問	悶	紋	門	匁							
ヤ		96DE:4C60:4464:										也	冶	夜	爺	耶	野	弥
		96EE:4C70:4480:	矢	厄	役	約	薬	訳	躍	靖	柳	薮	鑓					

	シフトJISコード JIS漢字コード 区点コード	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
ユ	96EE:4C70:4480:												愉	愈	油	癒	※
M	973F:4D20:4500:	※	諭	輸	唯	佑	優	勇	友	宥	幽	悠	憂	揖	有	柚	湧
	974F:4D30:4516:	涌	猶	猷	由	祐	裕	誘	遊	邑	郵	雄	融	夕			
ヨ	974F:4D30:4516:														予	余	与
	975F:4D40:4532:	誉	輿	預	傭	幼	妖	容	庸	揚	揺	擁	曜	楊	様	洋	溶
	976F:4D50:4548:	熔	用	窯	羊	耀	葉	蓉	要	謡	踊	遥	陽	養	慾	抑	欲
	9780:4D60:4564:	沃	浴	翌	翼	淀											
ラ	9780:4D60:4564:						羅	螺	裸	来	莱	頼	雷	洛	絡	落	酪
	9790:4D70:4580:	乱	卵	嵐	欄	濫	藍	蘭	覧								
リ	9790:4D70:4580:									利	吏	履	李	梨	理	璃	※
N	979E:4E20:4600:	※	痢	裏	裡	里	離	陸	律	率	立	葎	掠	略	劉	流	溜
	97AE:4E30:4616:	琉	留	硫	粒	隆	竜	龍	侶	慮	旅	虜	了	亮	僚	両	凌
	97BE:4E40:4632:	寮	料	梁	涼	猟	療	瞭	稜	糧	良	諒	遼	量	陵	領	力
	97CE:4E50:4648:	緑	倫	厘	林	淋	燐	琳	臨	輪	隣	鱗	麟				
ル	97CE:4E50:4648:													瑠	塁	涙	累
	97DE:4E60:4664:	類															
レ	97DE:4E60:4664:		令	伶	例	冷	励	嶺	怜	玲	礼	苓	鈴	隷	零	霊	麗
	97EE:4E70:4680:	齢	暦	歴	列	劣	烈	裂	廉	恋	憐	漣	煉	簾	練	聯	※
O	983F:4F20:4700:	※	蓮	連	錬												
ロ	983F:4F20:4700:					呂	魯	櫓	炉	賂	路	露	労	婁	廊	弄	朗
	984F:4F30:4716:	楼	榔	浪	漏	牢	狼	籠	老	聾	蝋	郎	六	麓	禄	肋	録
	985F:4F40:4732:	論															
ワ	985F:4F40:4732:		倭	和	話	歪	賄	脇	惑	枠	鷲	亙	亘	鰐	詫	藁	蕨
	986F:4F50:4748:	椀	湾	碗	腕																								
	9880:4F60:4764:																																
	9890:4F70:4780:																															※

A.3.4　コード間の変換

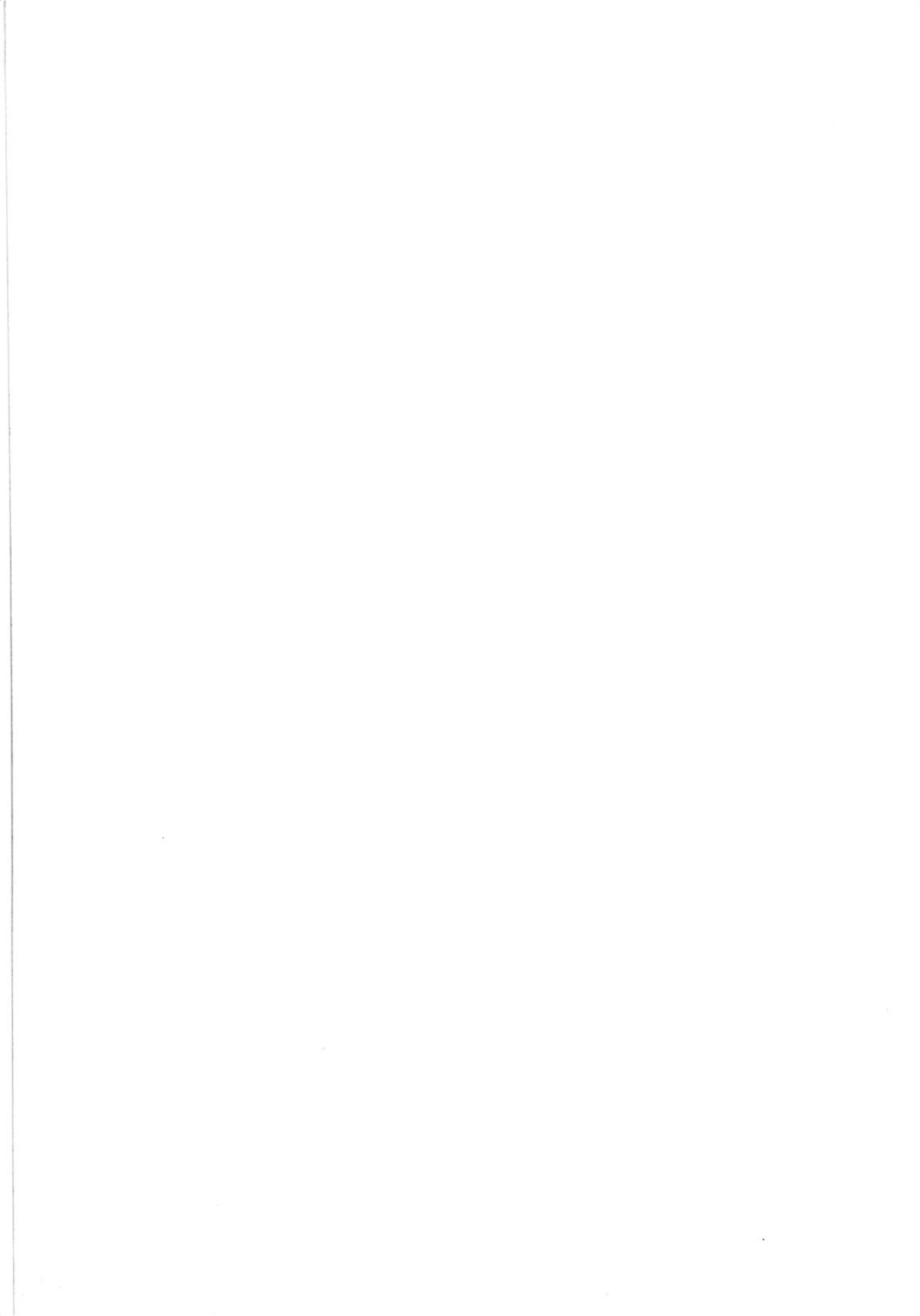
1バイトコードとキャラクタコードは次の関係式を用いて変換することができます。ここで、1バイトコード文字1文字をｘ＄と表わすことにします。

2バイトコード文字とシフトＪＩＳコード、区点コード、ＪＩＳ漢字コードの間は次の関係を用いて変換することができます。ここで、2バイトコード文字1文字をｙ＄と表わすことにします。

シフトＪＩＳコード
①
②
ＨＥＸ＄
（ＡＳＣ（ｙ＄））
ＣＨＲ＄（＆Ｈｎｎｎｎ）
⑥
ＪＩＳ＄（ｙ＄）
⑤
2バイトコード文字ｙ＄
ＪＩＳ漢字コード
ＣＨＲ＄（＆Ｊｎｎｎｎ）
ＣＨＲ＄
（＆Ｋｎｎｎｎ）
ＫＴＮ＄（ｙ＄）
③
区点コード
④

① ＪＩＳ＄（ＣＨＲ＄（＆Ｈｎｎｎｎ））
② ＨＥＸ＄（＆Ｊｎｎｎｎ）
③ ＫＴＮ＄（ＣＨＲ＄（＆Ｊｎｎｎｎ））
④ ＪＩＳ＄（ＣＨＲ＄（＆Ｋｎｎｎｎ））
⑤ ＫＴＮ＄（ＣＨＲ＄（＆Ｈｎｎｎｎ））
⑥ ＨＥＸ＄（＆Ｋｎｎｎｎ）

A.4 エラーメッセージ一覧表

エラーコード	エラーメッセージ	説明
1	NEXT without FOR	NEXT はあるのですが FOR がありません。
2	Syntax error	文法がまちがっています。
3	RETURN without GOSUB	GOSUB がないのに RETURN があります。
4	Out of data	READ で読むべきデータが DATA 文に用意されていません。
5	Illegal function call	規定外の数値やデータが指定されています。
6	Overflow	演算結果が許容範囲を越えました。
7	Out of memory	プログラムが大きすぎます。配列などの変数を多くとりすぎています。
8	Undefined label	GOTO、GOSUB、IF などで指定した分岐先の行番号がありません。
9	Subscript out of range	配列変数の添字が規定外です。
10	Duplicate definition	配列が2重に定義されています。
11	Division by zero	0では割れません。
12	Illegal direct	ダイレクト実行できないステートメンを実行しようとしました。
13	Type mismatch	変数の型が一致しません。
14	―――	―――――
15	String too long	文字が255文字を越えています。
16	Too complex	式が複雑すぎます。たとえば、( ) が異常に多い場合。
17	Can't continue	CONT によってプログラムの実行を再開できません。
18	Undefined function	DEF で定義されていない関数を呼びました。
19	No RESUME	RESUME によってプログラムの実行しようとしました。
20	RESUME without error	エラーがないのに RESUME を実行しようとしました。
21	―――	―――――
22	Missing operand	パラメータの必要な命令に指定がありません。
23	Line buffer overflow	1行の入力文字が多すぎます。
24	―――	―――――
25	Bad screen mode	グラフィックメモリを外部記憶として使おうとしました。
26	UNTIL without REPEAT	REPEAT がないのに UNTIL は実行できません。
27	Out of tape	カセットテープがセットされていません。
28	―――	―――――
29	Tape read error	カセットテープからデータが正しく読めません。
30	Bad file mode	異なったモードのファイルを参照しようとしました。(ファイルにはバイナリィ、中間コード、アスキータイプの3つの形式があります)
31	Out of stack	POP を実行しようとしたのですがスタッフに何も入っていません。

エラーコード	エラーメッセージ	説　　明
32	WHILE without WEND	WHILE ループに WEND がありません。
33	WEND without WHILE	WHILE がないのに WEND があります。
34	Reserved feature	将来の為のコマンドを使用しようとしました。
35	FOR without NEXT	FOR ループにNEXT がありません。
36	Format over	PRINT USING で指定したフォーマットが長すぎて出力できません。
37	REPEAT without UNTIL	REPEAT ループに UNTIL がありません。
50	FIELD overflow	FIELD 文でランダムファイル内のレコード長が256以上になっています。
51	Device in use	外部装置の使用中です。
52	Bad file number	オープンされていないファイルや、起動時に指定しないファイルを参照しようとしました。
53	File not found	LOAD、KILL、OPEN でディスクにないファイルを参照しようとしました。
54	Already open	すでに OPEN しているファイルを再び OPEN しようとしました。
55	―――	―――――
56	Device I/O error	入出力装置において入出力エラーが生じました。
57	File already exists	NAME で変更しようとしたファイル名はすでに登録されています。
58	―――	―――――
59	―――	―――――
60	Device full	データが入出力装置の許容容量を越えました。
61	Input past end	end of file のファイルを読もうとしたか、空ファイルを読もうとしました。
62	―――	―――――
63	―――	―――――
64	Bad allocation table	フロッピーディスク中のFAT が壊れています。
65	Bad file descripter	ディスクリプタが違います。
66	Bad record	レコード番号が規定外です。
67	No password	パスワードがあいません。
68	―――	―――――
69	―――	―――――
70	―――	―――――
71	File not open	OPEN されていないファイルを使用しようとしました。
72	Write protected	指定されたカセットテープ及びフロッピーディスクの消去防止用のプロテクトがされています。
73	Device offline	入出力装置がつながっていないのに使おうとしました。

注）ここに登録されていない表中の―――のエラー及び上記以外のエラーメッセージは、Unprintable Error と表示されます。

シャープ株式会社

本　　　　社　〒545　大阪市阿倍野区長池町22番22号
電話　06（621）1221（大代表）

電子機器事業本部　〒329-21　栃木県矢板市早川町１７４番地
電話　02874（3）1131（大代表）

お客様へ……お買いあげ年月日、お買いあげ店名を記入されますと、修理などの依頼のときに便利です。

お買いあげ年月日	年　　月　　日
お買いあげ店名	
	電話番号
もよりの お客様ご相談窓口	
	電話番号

TMAN-3615CE
T0796-A ④